Key States

Yuri Schaeffer, Yuri@NLnetLabs.nl

November 24, 2010

Contents

1 Key State

As Matthijs pointed out the state of a key can be described with individual states for each component (DS, DNSKEY, RRSIG). Since these are all resource records I hope to use a common state diagram for all types.

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 In the Withdrawn W state not all caches have the record or are able to get it.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W Withdrawn, some caches are unable to get the record.

2 State Transition

Let $\mathbb{R}$ be the set of possible resource records, according to the RFCs. Note that this includes all possible records, including many that make no sense to any zone owner anywhere.

Model a cache as the set of resource records from $\mathbb{R}$ that it currently holds. The type of the cache is $C : \mathbb{P}(\mathbb{R})$. We will assume that every cache adheres to the timing constraints imposed upon it by RFCs. These constraints cannot be made explicit in this model, because this model captures the (possible) situation(s) at a certain point in time. A more elaborate model, such as a Petri net, may add timing and thus capture the additional detail.

Caches. It is particularly useful to model a set $\mathbb{C} \subseteq \mathbb{P}(\mathbb{R})$ of all possible caches around the World, as this represents all possible states that could challenge a

[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.

$\emptyset$	Empty set	$\emptyset = \{\}$
$X \cup Y$	Union between two sets	$\{1,2\} \cup \{2,3\} = \{1,2,3\}$
$X \cap Y$	Intersection of two sets	$\{1,2\} \cap \{2,3\} = \{2\}$
$X \setminus Y$	Set subtraction	$\{1,2\} \setminus \{2,3\} = \{1\}$
$\cup X$	Union iterated over set elements	$\cup \{X,Y,Z\} = X \cup Y \cup Z$
$\cap X$	Intersection over set elements	$\cap \{X,Y,Z\} = X \cap Y \cap Z$
$\mathbb{P}(X)$	Powerset, or set of all subsets	$\mathbb{P}(\{1,2\}) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$

Figure 1: Mathematical notation used

DNSSEC signer to behave properly. Again, this model will not capture the timing constraints that define precisely what subset of $\mathbb{P}(\mathbb{R})$ actually forms $\mathbb{C}$.

Based on these definitions, we can derive a meaningful partition of the resource records $\mathbb{R}$ depending on how widely it is published:

$$\begin{aligned} \mathit{Hidden} &= \mathbb{R} \setminus \cup\mathbb{C} && (1)\\ \mathit{Propagating} &= \cup\mathbb{C} \setminus \cap\mathbb{C} && (2)\\ \mathit{Omnipresent} &= \cap\mathbb{C} && (3) \end{aligned}$$

Setting a goal. The above partition of $\mathbb{R}$ suffices to know what can be relied upon from the caches in the World. But, since DNS is not a static system, this state will evolve with time. Specifically, the *Propagating* state is a temporary one that will be departed as soon as the right TTL timing has passed. This however, can go either way; it depends on a new model element which way. This model element is the goal set for a resource record: Does it want to be *Omnipresent* or *Hidden*?

We therefore introduce another set named $\mathit{Announced} \subseteq \mathbb{R}$ to hold those resource records whose goal it is to end up in *Omnipresent*. Based on this, it is possible to partition *Propagating* even further:

$$\begin{aligned} \mathit{Rumoured} &= \mathit{Propagating} \cap \mathit{Announced} && (4)\\ \mathit{Squashed} &= \mathit{Propagating} \setminus \mathit{Announced} && (5) \end{aligned}$$

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.

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

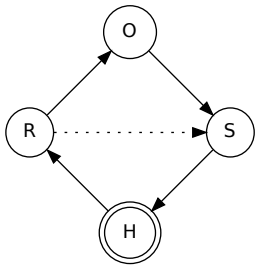

Figure 2: State diagram for individual key components.

2.1 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:

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

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

2.2 Transition rules for $Ds(k)$

2.2.1 $H(Ds(k))$

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.

$$\left.\begin{array}{l} Goal(k) = O \wedge ksk \notin Roles(k) \\ \\ Goal(k) = O \wedge O(Dnskey(k)) \\ \\ Goal(k) = O \wedge \exists k' \in \mathbb{K}(\\ \qquad Alg(k) = Alg(k') \\ \qquad \wedge ksk \in Roles(k') \\ \qquad \wedge O(Ds(k')) \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k'))) \end{array}\right\} \rightarrow [submit], R(Ds(k)) \tag{6}$$

2.2.2 $R(Ds(k))$

Nothing should depend on this RR yet.

$$Goal(k) = H \rightarrow S(Ds(k)) \tag{7}$$

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

$$\left.\begin{aligned} &Goal(k) = O \wedge ksk \notin Roles(k) \\ &Goal(k) = O \wedge Confirmed(k_{ds}) \\ &\qquad\qquad \wedge T_{now} \geq T_{whatever} \end{aligned}\right\} \rightarrow O(Ds(k)) \tag{8}$$

Postpone for later.

$$\begin{aligned} Goal(k) = O &\wedge Confirmed(k_{ds}) \\ &\wedge T_{now} < T_{whatever} \\ &\rightarrow [event] \end{aligned} \tag{9}$$

2.2.3 $O(Ds(k))$

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).

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

$$
\left.
\begin{aligned}
Goal(k) = H \wedge ksk \in Roles(k) \\
\wedge \forall r \in Roles(k) \\
(\\
\forall k' \in \mathbb{K}(k' \neq k \\
\wedge Alg(k) = Alg(k') \\
\wedge r \in Roles(k') \\
\rightarrow H(Ds(k'))) \\
\wedge \\
\exists k' \in \mathbb{K}(k' \neq k \\
\wedge r \in Roles(k') \\
\wedge O(Ds(k')) \\
\wedge O(Dnskey(k')) \\
\wedge O(Rrsig(k'))) \\
)
\end{aligned}
\right\} \rightarrow S(Ds(k)) \tag{10}
$$

$$
\left.
\begin{aligned}
Goal(k) = H \wedge ksk \notin Roles(k) \\
\wedge \exists k' \notin \mathbb{K}(k' \neq k \\
Alg(k) = Alg(k') \\
\wedge ksk \in Roles(k') \\
\wedge \neg H(Dnskey(k')) \\
\wedge O(Rrsig(k')))
\end{aligned}
\right\}
$$

2.2.4 $S(Ds(k))$

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

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Ds(k)) \tag{11}$$

$$T_{now} < T_{whatever} \rightarrow [event] \tag{12}$$

2.3 Transition rules for k_{dnskey}

2.3.1 $H(Dnskey(k))$

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

$$\left.\begin{array}{l} Goal(k) = O \wedge O(Rrsig(k)) \\ \qquad \wedge zsk \in Roles(k) \\ \\ Goal(k) = O \wedge O(Rrsig(k)) \\ \qquad \wedge \exists k' \in \mathbb{K}(\\ \qquad Alg(k') = Alg(k) \\ \qquad \wedge zsk \in Roles(k') \\ \qquad \wedge O(Ds(k')) \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k'))) \\ \\ Goal(k) = O \wedge \forall r \in Roles(k) \\ \qquad (\\ \qquad \exists k' \in \mathbb{K}(k' \neq k \\ \qquad \wedge Alg(k') = Alg(k) \\ \qquad \wedge r \in Roles(k') \\ \qquad \wedge O(Ds(k')) \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k'))) \\ \qquad) \end{array}\right\} \rightarrow R(Dnskey(k)) \tag{13}$$

2.3.2 $R(Dnskey(k))$

Skip the propagated state. Nobody heard of it anyway.

$$Goal(k) = H \rightarrow S(Dnskey(k)) \tag{14}$$

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

$$Goal(k) = O \wedge T_{now} \geq T_{whatever} \rightarrow O(Dnskey(k)) \tag{15}$$

We did not wait long enough: Wait some more.

$$Goal(k) = O \wedge T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{16}$$

2.3.3 $O(Dnskey(k))$

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.

$$
\left.
\begin{aligned}
Goal(k) = H \wedge H(Ds(k)) \\
\wedge H(Rrsig(k))
\end{aligned}
\right\}
\rightarrow S(Dnskey(k)) \tag{16}
$$

$$
\left.
\begin{aligned}
Goal(k) = H \wedge \forall r \in Roles(k) \\
(\\
\exists k' \in \mathbb{K}(k' \neq k \\
\wedge Alg(k') = Alg(k) \\
\wedge r \in Roles(k') \\
\wedge O(Ds(k')) \\
\wedge O(Dnskey(k')) \\
\wedge O(Rrsig(k'))) \\
)
\end{aligned}
\right\}
\rightarrow S(Dnskey(k)) \tag{17}
$$

$$
\left.
\begin{aligned}
Goal(k) = H \wedge H(Ds(k)) \\
\wedge \forall r \in Roles(k) \\
(\\
\forall k' \in \mathbb{K}(k' \neq k \\
\wedge Alg(k') = Alg(k) \\
\wedge r \in Roles(k') \\
\rightarrow H(Dnskey(k'))) \\
\wedge \\
\exists k' \in \mathbb{K}(k' \neq k \\
\wedge r \in Roles(k') \\
\wedge O(Ds(k')) \\
\wedge O(Dnskey(k')) \\
\wedge O(Rrsig(k'))) \\
)
\end{aligned}
\right\}
\rightarrow S(Dnskey(k)) \tag{18}
$$

2.3.4 $S(Dnskey(k))$

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

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Dnskey(k)) \tag{18}$$

Else, schedule event.

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{19}$$

2.4 Transition rules for k_{rrsig}

2.3.1 $H(Rrsig(k))$

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

$$Goal(k) = O \rightarrow R(Rrsig(k)) \tag{20}$$

2.4.2 $R(Rrsig(k))$

Don't wait till signatures are propagated.

$$Goal(k) = H \rightarrow S(Rrsig(k)) \tag{21}$$

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

$$\left.\begin{array}{l} Goal(k) = O \wedge T_{now} \geq T_{whatever} \\ \\ Goal(k) = O \wedge AllowSmooth \\ \qquad \wedge \forall r \in Roles(k)(\\ \qquad \exists k' \in \mathbb{K}(\\ \qquad Alg(k') = Alg(k) \\ \qquad \wedge r \in Roles(k') \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k')) \\ \qquad)) \end{array}\right\} \rightarrow O(Rrsig(k)) \tag{22}$$

Not enough time passed to propagate signatures.

$$Goal(k) = O \wedge T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{23}$$

2.4.3 $O(Rrsig(k))$

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

$$\left.\begin{array}{l} Goal(k) = H \wedge H(Dnskey(k)) \\ \\ Goal(k) = H \wedge \forall r \in Roles(k) \\ \quad (\\ \quad \exists k' \in \mathbb{K}(k' \neq k \\ \quad \wedge Alg(k) = Alg(k') \\ \quad \wedge r \in Roles(k') \\ \quad \wedge O(Ds(k')) \\ \quad \wedge O(Dnskey(k')) \\ \quad \wedge O(Rrsig(k'))) \\ \quad) \end{array}\right\} \rightarrow S(Rrsig(k)) \tag{24}$$

2.4.4 $S(Rrsig(k))$

Dnskey is gone so no one needs these signatures.

$$\left.\begin{array}{r} H(Dnskey(k)) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Rrsig(k)) \tag{25}$$

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{26}$$