DNSSEC Key State Transitions

The Grand Unified Theory of Everything DNSSEC ;)

Yuri Schaeffer, Yuri@NLnetLabs.nl

November 30, 2010

Contents

1 Introduction

During a key roll over each involved key has a state. Matthijs Mekking pointed out that the state-machine 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 roll-over centered approach is chosen 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 boundaries 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 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.

Using a cache centered 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.

This point of view makes sense since the identities validating the zone are viewing it from a cache's perspective. We only have to make sure every possible view on the data is valid at any point in time.

2.1 Key States

A key has two pieces of public information which are represented by three resource records: DS, DNSKEY, and RRSIG. Each of which can be known to a different set of caches and thus require their own state machine (Figure 1). When we say a key has goal X we mean that it wants to move all three records to state X.

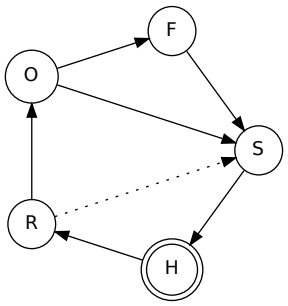

Figure 1: State diagram for individual records.

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 (anymore). 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 unexpired resource record set).

The two other states represent uncertainty: *Rumoured* and *Squashed*. The separation is a bit artificial as they more or less mean the same. Both represent that some caches do and some don't know about the resource record. However, in our model actions (which are observable by the outside world) are only taken on state transitions. If the goal of a key changes while a record is in an uncertain state the records reputation does not immediately change. We do however must explicitly tell the component responsible for publishing the record to introduce or withdraw it. This transition is represented by the dashed arrow in Figure 1.

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 *Squashed* 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. Or (Section 4.3) a sibling record.

***Squashed*:** The record is withdrawn but some caches might still have it.

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

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

2.2 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 $\mathbb{K}$. We can define DNSSEC validity over a keyring (see Section 3). It is required that every state transition of a resource record does not compromise the validity of the keyring as a whole.

In case a key's goal is to be *Hidden* and all three involved records are in state *Hidden*, the key may be removed from the keyring. Similar, if a key is in none of the keychains it could be purged from the keystore.

2.3 Key Operations

A key and its context can be described as a tuple of states $\langle Ds, Dnskey, Rrsig \rangle$ for the resource records DS, DNSKEY and RRSIG. We define the following operations on keys:

- $Ds(k)$, state of DS record belonging key k.
- $Dnskey(k)$, state of DNSKEY record belonging key k.
- $Rrsig(k)$, state of RRSIG record belonging key k.
- $Goal(k) \in \{Omnipresent, Hidden\}$, the state each record should move towards.
- $Roles(k) \subseteq \{zsk, ksk\}$ excluding $\emptyset$, the role(s) a key k should be used for.
- $Alg(k)$, algorithm of key k for which a transitive equivalence is defined.

3 DNSSEC Validity

With the definitions of Section 2.1 and beyond we can express the validity of a zone with respect to DNSSEC. After each state transition it must still hold. If not, the zone might be treated as *bogus* by some or all validators. This check does only takes the total set of states in to account, it has no notion of time. It can only guarantee every cache has a consistent view on the data when all transitions respected the time constraint and it was ran after each transition.

This check is not intended to be used by key management software but functions to validate our transition model, either empirical or by formal proof.

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

$$
\begin{aligned}
&ConsistentKeys \equiv \{k | k \in \mathbb{K}, \\
&\quad Roles(k) \subseteq ksk \rightarrow (\neg H(Ds(k)) \rightarrow O(Dnskey(k))) \wedge \\
&\quad \neg H(Dnskey(k)) \rightarrow O(Rrsig(k)) \wedge \\
&\quad (\\
&\qquad H(Dnskey(k)) \vee \\
&\qquad ksk = Roles(k) \rightarrow \exists k' \in \mathbb{K} \cdot (\\
&\qquad\quad zsk \in Roles(k') \wedge \\
&\qquad\quad Alg(k) = Alg(k') \wedge \\
&\qquad\quad O(Dnskey(k')) \wedge \\
&\qquad\quad O(Rrsig(k')) \\
&\qquad) \\
&\quad) \\
&\}
\end{aligned}
\tag{1}
$$

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

$$
\begin{aligned}
SafeKeys \equiv \{k | k \in \mathbb{K}, \\
&k \in ConsistentKeys \vee \\
&\forall r \in Roles(k) \cdot (\\
&\quad \exists k' \in \mathbb{K} \cdot (\\
&\quad\quad Alg(k') = Alg(k) \wedge \\
&\quad\quad r \in Roles(k') \wedge \\
&\quad\quad k' \in ConsistentKeys \wedge \\
&\quad\quad r = ksk \rightarrow (\neg H(Ds(k)) \rightarrow O(Ds(k'))) \wedge \\
&\quad\quad \neg H(Dnskey(k)) \rightarrow O(Dnskey(k')) \\
&\quad) \\
&) \\
\}
\end{aligned}
\tag{2}
$$

Validity A zone or keyring 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.

$$
\begin{aligned}
Valid(\mathbb{K}) \equiv \\
&\forall k \in \mathbb{K} \cdot k \in SafeKeys \wedge \\
&\exists k \in \mathbb{K} \cdot (\\
&\quad ksk \in Roles(k) \wedge \\
&\quad O(Ds(k)) \wedge \\
&\quad O(Dnskey(k)) \wedge \\
&\quad O(Rrsig(k)) \wedge \\
&\quad \exists k' \in \mathbb{K} \cdot (\\
&\quad\quad zsk \in Roles(k') \wedge \\
&\quad\quad O(Dnskey(k')) \wedge \\
&\quad\quad O(Rrsig(k')) \wedge \\
&\quad\quad Alg(k) = Alg(k') \\
&\quad) \\
&)
\end{aligned}
\tag{3}
$$

3.1 DNSSEC Validity (2)

The above rules can be greatly simplified under the assumption that a key in the DNSKEY set always also signs that set. Also, a having DS record implies a ksk.

$$
\begin{aligned}
&\exists k \in \mathbb{K} \cdot O(Ds(k)) \wedge \\
&\forall k \in \mathbb{K} \cdot (\\
&\quad (O(Dnskey(k)) \vee \\
&\quad \neg H(Ds(k)) \rightarrow \exists k' \in \mathbb{K} \cdot (\\
&\qquad Alg(k) = Alg(k') \wedge \\
&\qquad O(Ds(k')) \wedge \\
&\qquad O(Dnskey(k')) \wedge \\
&\qquad ksk \in Roles(k'))) \\
&\quad \wedge \\
&\quad (zsk \in Roles(k) \wedge O(Rrsig(k)) \vee \\
&\quad \neg H(Dnskey(k)) \rightarrow \exists k' \in \mathbb{K} \cdot (\\
&\qquad Alg(k) = Alg(k') \wedge \\
&\qquad O(Dnskey(k')) \wedge \\
&\qquad O(Rrsig(k')) \wedge \\
&\qquad zsk \in Roles(k'))))
\end{aligned}
\tag{4}
$$

4 Additional Considerations

4.1 Couple "RRSIG DNSKEY" with DNSKEY State

It would be better to couple the signature over the DNSKEY set with the state of the DNSKEY. There is really no use case to publish these seperately, plus they always travel simultaniously. As a consequence a combined key can describe to sign the data in the zone but not the DNSKEY set and vice versa.

Currently for the sake of uniformity a *zsk* has a DS. I propose to introduce a distinction between ksk, zsk, and ksk+zsk.

rrsig is overloaded.

ksk: DS, DNSKEY

zsk: DNSKEY, RRSIG

ksk+zsk: DS, DNSKEY, RRSIG

4.2 Urges

minimize DS set
  minimize RRSIG count
  minimize RRSIG flux

4.3 Omnipresent Sets of Keys

Our model defines the cache availablility for each key related resource record. This intruduces a problem for the RRSG for a *zsk* which does not represent a single record but in fact a whole collection. In the simpelest situation signatures can be replaced by new signatures as an atomic operation. It is also valid however to sign a subset of the data with one key and the rest with another key (possibly n keys).

Partial signing with multiple keys is a valid situation and a feature of the current signer. This enables a smooth transition between keys, with as goal to

spread out the workload for a signer. As enforcer we don't have information which record sets are signed with which key (because we don't want to have too much state). To support a smooth transition while ensuring we are maintaining a valid zone, we allow a set of keys to be responsible for signing. One of which should be in omnipresent state. If a key is part of this set it can move to the Omnipresent state for free, but it adds a restriction for leaving that set.

4.4 5011 Hack

extra F state between O and S. Conditions $O \to S \equiv O \to F + F \to S$. O-F: set revoke bit. F-S: wait till bit propagated.

5 Transition Rules

The transition rules are explicitly written in such a way that at least one valid chain is being kept; a zone can not go insecure[1]. To support the insecure concept one could introduce a NULL key. The NULL key has it's unique (NULL-)algorithm. Its resource records have state but no actual public data. This key can be rolled in and out like any other key.

5.1 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.

$$\left.\begin{array}{l} Goal(k) = O \wedge ksk \notin Roles(k) \\ \\ Goal(k) = O \wedge O(Dnskey(k)) \\ \\ Goal(k) = O \wedge \exists k' \in \mathbb{K}(\\ \qquad Alg(k) = Alg(k') \\ \qquad \wedge\, ksk \in Roles(k') \\ \qquad \wedge\, O(Ds(k')) \\ \qquad \wedge\, O(Dnskey(k')) \\ \qquad \wedge\, O(Rrsig(k'))) \end{array}\right\} \to [submit], R(Ds(k)) \tag{5}$$

[1] This prevents the system taking an 'insecure shortcut' to a new key.

$R(Ds(k))$

Nothing should depend on this RR yet.

$$Goal(k) = H \rightarrow S(Ds(k)) \tag{6}$$

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

$$\left.\begin{array}{r} Goal(k) = O \wedge ksk \notin Roles(k) \\ \\ Goal(k) = O \wedge Confirmed(k_{ds}) \\ \wedge T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow O(Ds(k)) \tag{7}$$

Postpone for later.

$$\begin{array}{r} Goal(k) = O \wedge Confirmed(k_{ds}) \\ \wedge T_{now} < T_{whatever} \\ \rightarrow [event] \end{array} \tag{8}$$

$O(Ds(k))$

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).

$$
\left.
\begin{aligned}
&Goal(k) = H \wedge \forall r \in Roles(k) \\
&\quad (\\
&\quad \exists k' \in \mathbb{K}(k' \neq k \\
&\quad \wedge Alg(k) = Alg(k') \\
&\quad \wedge r \in Roles(k') \\
&\quad \wedge O(Ds(k')) \\
&\quad \wedge O(Dnskey(k')) \\
&\quad \wedge O(Rrsig(k'))) \\
&\quad) \\
&\\
&Goal(k) = H \wedge ksk \in Roles(k) \\
&\quad \wedge \forall r \in Roles(k) \\
&\quad (\\
&\quad \forall k' \in \mathbb{K}(k' \neq k \\
&\quad \wedge Alg(k) = Alg(k') \\
&\quad \wedge r \in Roles(k') \\
&\quad \rightarrow H(Ds(k'))) \\
&\quad \wedge \\
&\quad \exists k' \in \mathbb{K}(k' \neq k \\
&\quad \wedge r \in Roles(k') \\
&\quad \wedge O(Ds(k')) \\
&\quad \wedge O(Dnskey(k')) \\
&\quad \wedge O(Rrsig(k'))) \\
&\quad) \\
&\\
&Goal(k) = H \wedge ksk \notin Roles(k) \\
&\quad \wedge \exists k' \notin \mathbb{K}(k' \neq k \\
&\quad Alg(k) = Alg(k') \\
&\quad \wedge ksk \in Roles(k') \\
&\quad \wedge \neg H(Dnskey(k')) \\
&\quad \wedge O(Rrsig(k')))
\end{aligned}
\right\} \rightarrow S(Ds(k)) \qquad (9)
$$

$S(Ds(k))$

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

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Ds(k)) \tag{10}$$

$$T_{now} < T_{whatever} \rightarrow [event] \tag{11}$$

5.2 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.

$$\left.\begin{array}{l} Goal(k) = O \wedge O(Rrsig(k)) \\ \qquad \wedge zsk \in Roles(k) \\ \\ Goal(k) = O \wedge O(Rrsig(k)) \\ \qquad \wedge \exists k' \in \mathbb{K}(\\ \qquad Alg(k') = Alg(k) \\ \qquad \wedge zsk \in Roles(k') \\ \qquad \wedge O(Ds(k')) \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k'))) \\ \\ Goal(k) = O \wedge \forall r \in Roles(k) \\ \qquad (\\ \qquad \exists k' \in \mathbb{K}(k' \neq k \\ \qquad \wedge Alg(k') = Alg(k) \\ \qquad \wedge r \in Roles(k') \\ \qquad \wedge O(Ds(k')) \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k'))) \\ \qquad) \end{array}\right\} \rightarrow R(Dnskey(k)) \tag{12}$$

$R(Dnskey(k))$

Skip the propagated state. Nobody heard of it anyway.

$$Goal(k) = H \rightarrow S(Dnskey(k)) \tag{13}$$

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

$$Goal(k) = O \wedge T_{now} \geq T_{whatever} \rightarrow O(Dnskey(k)) \tag{14}$$

We did not wait long enough: Wait some more.

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

$O(Dnskey(k))$

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.

$$
\left.
\begin{array}{l}
Goal(k) = H \wedge H(Ds(k)) \\
\qquad \wedge H(Rrsig(k)) \\
\\
Goal(k) = H \wedge \forall r \in Roles(k) \\
\qquad (\\
\qquad \exists k' \in \mathbb{K}(k' \neq k \\
\qquad \wedge Alg(k') = Alg(k) \\
\qquad \wedge r \in Roles(k') \\
\qquad \wedge O(Ds(k')) \\
\qquad \wedge O(Dnskey(k')) \\
\qquad \wedge O(Rrsig(k'))) \\
\qquad) \\
\\
Goal(k) = H \wedge H(Ds(k)) \\
\qquad \wedge \forall r \in Roles(k) \\
\qquad (\\
\qquad \forall k' \in \mathbb{K}(k' \neq k \\
\qquad \wedge Alg(k') = Alg(k) \\
\qquad \wedge r \in Roles(k') \\
\qquad \rightarrow H(Dnskey(k'))) \\
\qquad \wedge \\
\qquad \exists k' \in \mathbb{K}(k' \neq k \\
\qquad \wedge r \in Roles(k') \\
\qquad \wedge O(Ds(k')) \\
\qquad \wedge O(Dnskey(k')) \\
\qquad \wedge O(Rrsig(k'))) \\
\qquad)
\end{array}
\right\} \rightarrow S(Dnskey(k)) \tag{16}
$$

$S(Dnskey(k))$

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

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Dnskey(k)) \tag{17}$$

Else, schedule event.

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{18}$$

5.3 Transition rules for $Rrsig(k)$

$H(Rrsig(k))$

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

$$Goal(k) = O \rightarrow R(Rrsig(k)) \tag{19}$$

$R(Rrsig(k))$

Don't wait till signatures are propagated.

$$Goal(k) = H \rightarrow S(Rrsig(k)) \tag{20}$$

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

$$\left.\begin{array}{l} Goal(k) = O \wedge T_{now} \geq T_{whatever} \\ \\ Goal(k) = O \wedge AllowSmooth \\ \qquad \wedge \forall r \in Roles(k)(\\ \qquad \exists k' \in \mathbb{K}(\\ \qquad Alg(k') = Alg(k) \\ \qquad \wedge r \in Roles(k') \\ \qquad \wedge O(Dnskey(k')) \\ \qquad \wedge O(Rrsig(k')) \\ \qquad)) \end{array}\right\} \rightarrow O(Rrsig(k)) \tag{21}$$

Not enough time passed to propagate signatures.

$$Goal(k) = O \wedge T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{22}$$

$O(Rrsig(k))$

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

$$\left.\begin{array}{l} Goal(k) = H \wedge H(Dnskey(k)) \\ \\ Goal(k) = H \wedge \forall r \in Roles(k) \\ \quad (\\ \quad \exists k' \in \mathbb{K}(k' \neq k \\ \quad \wedge Alg(k) = Alg(k') \\ \quad \wedge r \in Roles(k') \\ \quad \wedge O(Ds(k')) \\ \quad \wedge O(Dnskey(k')) \\ \quad \wedge O(Rrsig(k'))) \\ \quad) \end{array}\right\} \rightarrow S(Rrsig(k)) \tag{23}$$

$S(Rrsig(k))$

Dnskey is gone so no one needs these signatures.

$$\left.\begin{array}{r} H(Dnskey(k)) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Rrsig(k)) \tag{24}$$

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{25}$$