DNSSEC Key State Transitions

The Grand Unified Theory of Everything DNSSEC ;)

Yuri Schaeffer, Yuri@NLnetLabs.nl

December 9, 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 its records to state X.

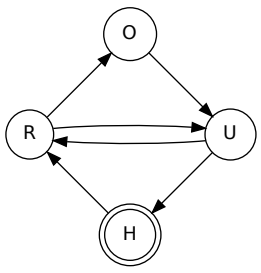

Figure 1: State diagram for individual records.

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 *Unretentive*. 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.

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, *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:

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

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. Only keys acting in the

role of ksk have a Ds state, similar only a zsk has a $Rrsig$ state. Furthermore we require the signature over the DNSKEY set to propagate together with the DNSKEY. We define the following operations on a key k:

$Ds(k)$ 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.

$Goal(k)$ $Goal(k) \in \{Omnipresent, Hidden\}]$ the state each record should move towards.

$Roles(k)$ $Roles(k) \subseteq \{zsk, ksk\}]$ excluding $\emptyset$, the role(s) a key k should be used for. $Ds(k) \wedge Dnskey(k) \equiv ksk \in Roles(k)$ and $Dnskey(k) \wedge Rrsig(k) \equiv zsk \in Roles(k)$

$Alg(k)$ Algorithm of key k for which a transitive and commutative 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 the validity 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.

$$
\begin{aligned}
&\exists k \in \mathbb{K} \cdot O(Ds(k)) \wedge \\
&\forall k \in \mathbb{K} \cdot (\\
&\quad (H(Ds(k)) \vee \\
&\quad O(Dnskey(k)) \vee \\
&\quad \exists k' \in \mathbb{K} \cdot (\\
&\qquad Alg(k) = Alg(k') \wedge \\
&\qquad O(Ds(k')) \wedge \\
&\qquad O(Dnskey(k')))) \\
&\quad \wedge \\
&\quad (H(Dnskey(k)) \vee \\
&\quad O(Rrsig(k)) \vee \\
&\quad \exists k' \in \mathbb{K} \cdot (\\
&\qquad Alg(k) = Alg(k') \wedge \\
&\qquad O(Dnskey(k')) \wedge \\
&\qquad O(Rrsig(k'))))) \qquad (1)
\end{aligned}
$$

4 Additional Considerations

Please ignore this section

4.1 5011 Hack

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

5 Rollover Scenarios

The proposed state machine has no notion of rollover scenarios. It cares only about the validity of the zone in terms of DNSSEC. Stricly following the transition rules results in the fastest possible transition. It is however sometimes desired to do a slower rollover in favor of the size of the data, traffic, computing power or external interaction. Current described rollover mechanisms can be described as a set of constraints on the previously mentioned quantities.

Important to realize is that different rollover mechanisms do not need different rules. All operate within the boundries of DNSSEC validity which is descibed by our model. A rollover is nothing more than a requirement to do certain transitions in a pre defined order. To achieve that, selective brakes have to be applied on the system.

Minimize Parent Interactions Rolling to a new KSK requires the submission of a new and withdrawing of an old DS record in the parent of the deligation. Child-Parent interaction may be slow, performing these two actions simultainiously could save time and efford.

Minimize Signatures Especially for large zones it may be desireable to prevent having more than one signature over each resource record set.

Minimize DNSKEY Set TODO

On top of this one might want to prevent replacing all signatures at once. A suddent transition may cause a peak load on the servers and can cause too much stress on the signer. This does not influence the transition rules but rather the interpretation of the states in generating a signer configuration.

$MinSig$ The system tries to mimimize the number of concurrent signatures by waiting till the DNSKEY is *Omnipresent*. Effectively pre and postpublishing the DNSKEY.

$MinFlux$ If set the mapping for the signer configuration will allow a smooth transition between keys. $MinFlux \rightarrow MinSig$.

5.1 Signer Configuration Mapping

Doing smooth transitions (MinFlux) between two keys requires us to know how the signer interprets the configuration we feed it. In a normal situation the state of a record translates directly to signer directives for that key.

$$
\begin{aligned}
Published(k) \equiv \quad & (R(Dnskey(k)) \vee O(Dnskey(k))) \wedge \\
& (zsk \notin Roles(k) \vee \\
& MinFlux \vee \\
& O(goal(k)) \vee \\
& \exists k' \notin \mathbb{K} \cdot (\\
& \quad O(Dnskey(k')) \wedge \\
& \quad (O(Rrsig(k')) \vee R(Rrsig(k'))) \wedge \\
& \quad O(Goal(k')) \wedge \\
& \quad Alg(k) = Alg(k')))
\end{aligned} \tag{2}
$$

$$
\begin{aligned}
Active(k) \equiv \quad & (R(Rrsig(k)) \vee O(Rrsig(k))) \wedge \\
& zsk \in Roles(k) \wedge \\
& (O(goal(k)) \vee \\
& \exists k' \notin \mathbb{K} \cdot (\\
& \quad O(Dnskey(k')) \wedge \\
& \quad (O(Rrsig(k')) \vee R(Rrsig(k'))) \wedge \\
& \quad O(Goal(k')) \wedge \\
& \quad Alg(k) = Alg(k')))
\end{aligned} \tag{3}
$$

6 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.

6.1 Transition Rules for $Ds(k)$

Transition rules for Ds states are only defined for KSKs. Furthermore a Ds state of key k implies that k has the role of KSK. A premise involving the Ds state of a non-KSK key is always false.

6.1.1 $H(Ds(k))$

The DS record is hidden, available in no cache whatsoever. Before submitting either the DNSKEY must be propagated or an other key with the same algorithm must be ready.

$$
\left.
\begin{aligned}
& O(Goal(k)) \wedge \\
& \quad (O(Dnskey(k)) \vee \\
& \quad \exists k' \in \mathbb{K} \cdot (\\
& \qquad Alg(k) = Alg(k') \wedge \\
& \qquad O(Ds(k')) \wedge \\
& \qquad O(Dnskey(k'))))
\end{aligned}
\right\} \vdash [submit], R(Ds(k)) \tag{4}
$$

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

6.1.2 $R(Ds(k))$

Because the DS record was never fully known in the world (at least to our own knowledge) we can savely withdraw it. Nothing started to depend on it yet.

$$H(Goal(k)) \vdash U(Ds(k)) \tag{5}$$

This transition depend on external processes, possibly human. If it is confirmed the process was successfull and enough time has passed since then, we may conclude the DS record is now *Omnipresent*.

$$\left.\begin{array}{l} O(Goal(k)) \wedge \\ Confirmed(k_{ds}) \wedge \\ T_{now} \geq T_{whatever} \end{array}\right\} \vdash O(Ds(k)) \tag{6}$$

6.1.3 $O(Ds(k))$

We may withdraw a DS record if there is another valid chain and we do not risk breaking another chain of the same algorithm.

$$\left.\begin{array}{l} H(Goal(k)) \wedge \\ \exists k' \in \mathbb{K} \cdot (\\ \quad k' \neq k \wedge \\ \quad O(Ds(k'))) \wedge \\ \forall k' \in \mathbb{K} \cdot (\\ \quad Alg(k) \neq Alg(k') \vee \\ \quad H(Ds(k')) \vee \\ \quad O(Dnskey(k')) \vee \\ \quad \exists k'' \in \mathbb{K} \cdot (\\ \quad\quad Alg(k'') = Alg(k') \wedge \\ \quad\quad O(Ds(k'')) \wedge \\ \quad\quad O(Dnskey(k'')) \wedge \\ \quad\quad k'' \neq k)) \end{array}\right\} \vdash [withdraw], U(Ds(k)) \tag{7}$$

6.1.4 $U(Ds(k))$

Again, without punishment we can move between the two uncertain states.

$$O(Goal(k)) \vdash [submit], R(Ds(k)) \tag{8}$$

We must wait till at least $T_{whatever}$ before transition to State *Hidden*.

$$T_{now} \geq T_{whatever} \vdash H(Ds(k)) \tag{9}$$

6.2 Transition rules for $Dnskey(k)$

Every key, independent of role, has a DNSKEY record. Therefore we must be extra carefull about the type of key the record belongs to.

6.2.1 $H(Dnskey(k))$

A KSK can only introduce a DNSKEY record if there is a ZSK of the same algorithm ready. A ZSK may additionally introduce it if its RRSIG is *Omnipresent*.

$$\left.\begin{array}{l} O(Goal(k)) \wedge \\ \quad (O(Rrsig(k)) \vee \\ \quad \exists k' \in \mathbb{K} \cdot (\\ \qquad Alg(k) = Alg(k') \wedge \\ \qquad O(Dnskey(k)) \wedge \\ \qquad O(Rrsig(k)))) \end{array}\right\} \vdash [submit], R(Dnskey(k)) \tag{10}$$

6.2.2 $R(Dnskey(k))$

We can jump freely between uncertain states.

$$H(Goal(k)) \vdash [withdraw], U(Dnskey(k)) \tag{11}$$

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

$$O(Goal(k)) \wedge T_{now} \geq T_{whatever} \vdash O(Dnskey(k)) \tag{12}$$

6.2.3 $O(Dnskey(k))$

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.

$$
\left.
\begin{array}{l}
H(Goal(k)) \wedge \\
(\mathbf{zsk} \notin \mathbf{Roles(k)} \vee \\
\quad \neg \mathbf{MinSig} \vee \\
\quad \mathbf{H(Rrsig(k))} \vee \\
\quad \exists \mathbf{k'} \notin \mathbb{K} \cdot (\\
\quad\quad \mathbf{H(Rrsig(k'))} \wedge \\
\quad\quad \mathbf{Alg(k) = Alg(k'))}) \wedge \\
\forall k' \in \mathbb{K} \cdot (\\
\quad (ksk \notin Roles(k') \vee \\
\quad H(Ds(k')) \vee \\
\quad (O(Dnskey(k')) \wedge \\
\quad\quad k \neq k') \vee \\
\quad \exists k'' \in \mathbb{K} \cdot (\\
\quad\quad k \neq k'' \wedge \\
\quad\quad Alg(k') = Alg(k'') \wedge \\
\quad\quad O(Ds(k'')) \wedge \\
\quad\quad O(Dnskey(k'')))) \\
\quad \wedge \\
\quad (H(Dnskey(k')) \vee \\
\quad O(Rrsig(k')) \vee \\
\quad \exists k'' \in \mathbb{K} \cdot (\\
\quad\quad k \neq k'' \wedge \\
\quad\quad Alg(k') = Alg(k'') \wedge \\
\quad\quad O(Dnskey(k'')) \wedge \\
\quad\quad O(Rrsig(k''))))
\end{array}
\right\} \vdash [withdraw], U(Dnskey(k)) \tag{13}
$$

6.2.4 $U(Dnskey(k))$

We can jump freely between uncertain states.

$$Goal(K) = O \vdash [submit], R(Dnskey(k)) \tag{14}$$

State may transition to $Hidden$ given enough time passed to propagate change.

$$T_{now} \geq T_{whatever} \vdash H(Dnskey(k)) \tag{15}$$

6.3 Transition rules for $Rrsig(k)$

6.3.1 $H(Rrsig(k))$

Signatures may be introduced at any possible time.

$$
\left.
\begin{array}{l}
O(Goal(k)) \wedge \\
(\neg \mathbf{MinSig} \vee \\
\quad \mathbf{O(Dnskey(k))} \vee \\
\quad \exists \mathbf{k'} \notin \mathbb{K} \cdot (\\
\quad\quad \mathbf{O(Dnskey(k'))} \wedge \\
\quad\quad \mathbf{Alg(k) = Alg(k'))} \vdash [submit], R(Rrsig(k))
\end{array}
\right\} \vdash U(Rrsig(k)) \tag{16}
$$

6.3.2 $R(Rrsig(k))$

We can jump freely between uncertain states.

$$H(Goal(k)) \vdash [withdraw], U(Rrsig(k)) \tag{17}$$

Enough time passed to know for sure the signatures are propagated.

$$O(Goal(k)) \wedge T_{now} \geq T_{whatever} \vdash O(Rrsig(k)) \tag{18}$$

6.3.3 $O(Rrsig(k))$

If the dnskey is gone from all caches can we withdraw the signatures safely. In any other case a fully propagated ZSK will also do.

$$\left.\begin{array}{l} H(Goal(k)) \wedge \\ \quad (H(Dnskey(k)) \vee \\ \quad \exists k' \in \mathbb{K} \cdot (\\ \qquad k' \neq k \wedge \\ \qquad Alg(k) = Alg(k') \wedge \\ \qquad O(Dnskey(k')) \wedge \\ \qquad O(Rrsig(k')))) \end{array}\right\} \vdash U(Rrsig(k)) \tag{19}$$

6.3.4 $U(Rrsig(k))$

We can jump freely between uncertain states.

$$O(Goal(k)) \vdash [submit], R(Rrsig(k)) \tag{20}$$

After some time we know this signatures are no longer know to the world.

$$T_{now} \geq T_{whatever} \vdash H(Rrsig(k)) \tag{21}$$