Flexible and Robust Key Rollover in DNSSEC

Yuri Schaeffer, Benno Overeinder, and Matthijs Mekking
NLnet Labs, Amsterdam, The Netherlands
Email: {yuri,benno,matthijs}@NLnetLabs.nl

***Abstract*—DNSSEC security extensions make use of a public-private key pair to sign and validate origin and integrity of DNS data. The ability to renew keys is a standard operational practice in the deployment of DNSSEC. This key renewal, or actually key rollover, is a complex and error prone process. We propose a new method for key rollover in which not the individual *procedural* steps of a rollover are specified, but the *validity* of a step in the rollover process is specified. The rollover process can now find an optimal and correct path from an old key to a new key. The proposed method is robust, is effective in emergency situations in which a compromised key must be rolled over in the shortest amount of time possible, and allows for efficient combined rollover of multiple keys. The new key rollover method presented in this paper is implemented and integrated within the OpenDNSSEC software framework.**

I. Introduction

The Domain Name System (DNS) [1] is an infrastructure that allows the usage of human readable names to address hosts and services on the Internet. As such, DNS is considered a critical infrastructure for many uses and users, like in finance, businesses, governments, news, and social media.

One of the largest threats to DNS dependability is the injection of fake (incorrect) responses by DNS cache poisoning [2], [3]. In the past, various patches have been applied to improve the security of DNS and to counter the threats discovered. To end the introduction of ad-hoc solutions to new threats, a more fundamental solution to secure the DNS protocol has been developed. DNS Security Extensions (DNSSEC) [4] is a set of extensions to DNS which provide data origin authentication, data integrity, and authenticated denial of existence. With these extensions, it is possible to detect fake responses to DNS queries.

DNSSEC works by digitally signing DNS data using public-key cryptography. The public key of the DNS root zone is well-known and used to sign data and other keys, which in turn can be used to sign yet other data and keys, and so on to an arbitrary depth. The correctness of signed DNS records can now be authenticated via an *authentication chain*, starting with the set of verified public keys for the DNS root zone.

As with any public-key system, the trust in a key decreases over time. The more often a key is used, the higher the chance the key material is compromised by some means, e.g., by reverse engineering the private key. To maintain the integrity of the system, individual keys in the chain need to be renewed. Operational reasons to renew keys are due to new hardware, new software, new employees, new parameters, and registrar exercises for emergency situations [5], [6], [7]. Unfortunately, renewing keys is a complex, operational task as DNS heavily relies on distributed caching of information. If one would simply renew keys by switching from old to new, the validation process could have a mixed view on the data (e.g., a new key but old signatures) and declare the zone invalid. Thus rather than switched, keys must be *rolled*.

The key rollover procedure can be realized in many ways, but the correct order of operations is important. DNSSEC is not forgiving, and an incorrect order of rollover operations may result in a loss of trust in all derivative keys. Putting differently, there are various transition paths to roll over a key, but only some of them are valid transition paths.

The common and most straight forward approach is a procedural specification of a set of possible correct rollovers to roll a key A to a new key B. Preferably, this set of specified rollovers is relatively small, as each different rollover would require its own procedure to realize a valid transition path. We propose an alternative approach that defines a set of valid key states and goals that have to be realized in the rollover of a key. Instead of specifying a set of specific rollover transition paths, we specify the conditions whether a transition results in a valid key state. The system will find the shortest safe path towards a desired state. In principle, this allows the key rollover process to find arbitrary valid state transition paths for one or more keys combined. The potential of concurrent key rollovers is of large importance for emergency situations, where a compromised key has to be replaced as soon as possible, even if an ongoing key rollover has to be aborted.

In this paper, some background on DNSSEC and key rollover is presented in Section II. Next, the main ideas and concepts behind valid key states and their transitions are presented in Section III. In Section IV, the concepts are combined and formal rules about the validity of a set of keys are specified. An implementation of the new approach will be part of the OpenDNSSEC[1] software and some key rollover traces are shown, see Section V. Finally, Section VI concludes the paper with some future work and perspectives.

II. Background

The main goal of DNSSEC is to protect against data corruption [8]. While the intricacies of DNSSEC go far beyond the scope of the paper, some specific context will be presented related to key states and rollover.

The client-side of the DNS is called a resolver. It is responsible for initiating and sequencing the queries that eventually results in the full resolution of a resource, e.g., a domain name

[1] http://www.opendnssec.org/

into an IP address. A resolver that implements DNSSEC, a *validator*, will have to determine the security status of data received in responses. Such resolvers, are configured with a trust anchor, preferably the root key set, from which it can build a *chain of trust* to authenticate zone data. If the resolver is able to build such an authentication chain from the trust anchor to the data, the status of that data will be marked as *secure*. If this chain is known to exist, but cannot be constructed, the data will be considered as *bogus*.

DNSSEC adds new records to an existing DNS zone. The DNSKEY record holds a public key and the cryptographic algorithm (e.g., SHA-1, SHA-256, etc.) that is used with the key. The signatures created with the key are stored in RRSIG records. The DS record helps in constructing an authentication chain and is put in the upper, parental side of a delegation. With a hash, it points to a DNSKEY record in the delegated child zone. This is how DNSSEC secures delegations. Furthermore, NSEC and NSEC3 records are added to provide authenticated denial of existence, but these records are less relevant with respect to our new approach.

Records are stored as resource record sets (RRsets) in caches and may be validated again at a later point in time. If present, the signatures are attached to the RRset as RRSIG records. They are accompanied with a Time-To-Live (TTL), the maximum amount of time that a resolver can hold this data in the cache.

In order to validate an RRset, a validator needs to acquire the corresponding DNSKEY, which contains the public key. Usually, the validator is configured with the root trust anchor, and the required DNSKEY can be obtained through secure delegations. It is verified that the root trust anchor matches the DNSKEY at the apex of the zone and then the DS RRset can be used to validate secure delegations. The DS record identifies a DNSKEY in the child zone, which can be used to validate RRsets in that zone. This process is repeated for each intermediate delegation. Once the DNSKEY matching the signature is found, the data can be validated.

A. Key Types

Keys can be of different types, depending on their *role*. A *Key Signing Key* (KSK) is used to authenticate the zone signing keys. Such key is solely responsible for creating a signature for the DNSKEY RRset. A *Zone Signing Key* (ZSK) creates the signatures for all other RRsets and is used to authenticate the associated zone data. A key can have both roles at the same time, sometimes referred to as a *Combined Signing Key* (CSK).

B. Key Rollover

The ability of replacing keys is an important aspect of DNSSEC. With key rollover, the successor key will not be immediately visible to validating resolvers due to the distributed caching of keys. Also, data published in previous versions of the zone may still exist in caches. The timing of when to introduce and when to withdraw data from the zone is of vital essence. Using the wrong values for the timing parameters may break the chain of trust and cause validators to mark data as bogus, effectively making the zone invisible to clients on the Internet.

For example, if key a is being replaced with key b, the DNSKEY for key b should be pre-published first. Keeping both keys published for a sufficient period of time allows validators to gain knowledge of both public keys. Only after this period, signatures created with a can be replaced with the ones created by b. If not enough time has passed and some validators only know about key a, these validators might be unable to validate RRsets. Suppose an RRset that just expired from the cache. A new query is now being directed towards the authoritative name server, where signatures have been replaced recently. The validator now ends up with the public key for a and an RRset signed by b. Due to timing mistakes, this zone data can now no longer be validated. Only when the DNSKEY RRset expires, the new public key will be learned and the security status will be restored.

In other words, validators must always be able to build chains, regardless of whether the data came from the cache, originating name server, or an intermediate, forwarding system with its own cache. And the administrators of the secured zones are responsible for keeping the chains of trust intact. Current practice is to set up a procedural specification for the rollover, so that administrators can implement the process by following the specified steps [9]. Each step changes the properties of the keys involved. With these *key states*, one can track the progress of the rollover.

Although there exists similarity between rollovers, each specification describes a different set of transitions to be taken in order to correctly implement the process. Such document is usually very complex and with a great level of detail, and one can easily make mistakes when crafting, or implementing them. For that reason, it would be best to keep the set of specified rollovers as small as possible. However, zone administrators desire different flavours of rollovers. One procedure switches keys as fast as possible, while other rollovers focus on keeping the size of the zone and the response size to a minimum. Local policy determines whether an administrator will use one procedure or the other. A policy that is being updated may trigger additional "transit" rollovers. Another factor to consider is that the timings for ZSK rollovers differ from those for KSK rollovers. And similar differences hold for CSK rollovers, where keys have both roles at the same time.

In other words, there exist many different ways to replace keys, and each type of rollover requires its own elaborate procedural specification of the timing parameters. Creating a new type of rollover implies work to be done that is non-trivial and error prone. By this complexity, most rollover procedures are limited to replacing two keys, the operation of introducing one and withdrawing one key. A different approach is to look at rollovers being a process of replacing a *set of keys*. For example, when switching signing schemes, two keys (a KSK and a ZSK) are being replaced with just one CSK, or vice versa. With the current approach this would result in a very

large number of key replacement specifications.

Another limitation is that the rollover procedures have firmly defined the order of operations of key replacement, from start to end. As a consequence, a concurrent rollover strategy is not specified. Usually this is not a problem, as most rollovers are scheduled and would not intervene with each other. But in case of a key compromise, you would like to perform an emergency rollover, to disable the use of the key as fast as possible. With the current, procedural strategy, we might have to delay the emergent event until the rollover in progress has finished.

The complexity of the rollover procedures, in combination with those limitations, puts a high barrier on the innovation of key rollovers. An alternative and potentially more versatile approach can be attained by the dynamical *discovery* of the best possible rollover strategy that fits a policy.

III. Key States Unraveled

Instead of having a specification for each different rollover scenario, our approach focuses on whether a step in an arbitrary rollover results in a valid situation with respect to DNSSEC, and in effect by doing this, dynamically discover a valid order of key rollover operations. In order to make a clear distinction of what is a valid situation with respect to DNSSEC, some relevant concepts need to be defined.

A. A Cache Centric Approach

The common existing approach can be called rollover centric, and considers a rollover to be a procedural specification of multiple sequential steps with strict order and timing requirements, that describe the scenario from start to end. A *cache centric* approach looks at the data itself from the perspective of *all* caching validators in the global DNS infrastructure. More important, it is concerned about the validity of the data, and whether *all validators are able to build a chain of trust* with the available parts, regardless where the data comes from. This is the basic requirement in order for DNSSEC to work. Our approach tries to convert this requirement into formal rules. This way of thinking allows us to do everything we want within the boundaries of DNSSEC validity, and we are no longer limited to a sequential rollover strategy or to the number of keys involved.

B. A View On Keys

Though there exist many ways to roll a key, all mechanisms share the same principle: A set of existing keys needs to be replaced by a set of new keys within the boundaries of DNSSEC validity. Over time, all rollover transition paths will have to introduce the DNSSEC resource records related to the new keys, and withdraw the DNSSEC records related to the existing keys. Validators gain information about the key over time by querying for these records. This information consists of the public key, stored in the DNSKEY record, the corresponding secure delegation, the DS record, and its created signatures as RRSIGs. Because the resource records most probably are not published at the same time, caches can have an incomplete view of keys.

Key states in the current rollover approach determine the progress of the rollover. Instead of one complicate state machine per key, we propose to maintain a state machine for each key related resource record. This gives us four separate state machines, relating to the records DS, DNSKEY, RRSIG DNSKEY, and RRSIG. The latter does not represent a single record, but rather all signatures in the zone (except for the signature over the DNSKEY RRset). Because keys can either be a ZSK or KSK (or both), a distinction is needed between the signature for the DNSKEY RRset and the other signatures.

By *unraveling* the key states in such a way, we are able to model the incomplete view validators can have on keys more accurately. Each key now has its own set of interdependent machines, one for each key related record (see Figure 1).

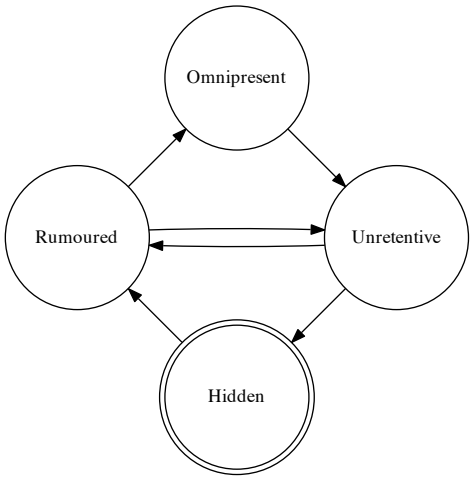

Fig. 1. State diagram for individual records.

These machines reflect the visibility of the resource records by all possible caches in the global DNS infrastructure. Therefore, all machines have the same set of states. The life-cycle of a resource record starts and ends in the *Hidden* state. In this state, none of the validators is able to observe this record. The opposite state is the *Omnipresent* state: all validators will have the record in cache or are able to refetch it when needed. The two other states represent the uncertainty in our model. In the *Rumoured* state, the record is published in the zone but not all caches will know about this record yet, while in the *Unretentive* state, the record has been removed from the zone but some caches might still have it in cache. In these two states, one cannot rely on this record as a dependency to build a chain of trust. It is perfectly safe to hop between the uncertain Rumoured and Unretentive states. These transitions allow us to prematurely end a rollover, undo a rollover, or do multiple rollovers in parallel.

C. Key Goals

Keys are used for a certain purpose. Either a key is going to be activated so that it can be used to perform authentication, or it is going to be removed to make it inoperable. When activating a key, all corresponding state machines will try to reach the Omnipresent state. When removing a key, the goal is to get all corresponding machines in the Hidden state. In other words, the *key goal* is either Omnipresent or Hidden.

The goal of the key has a direct influence to what transitions will be made. As long as the state is equal to the key goal, the record is said to be stable and will not try to go to another state. In an unstable situation, the machines will try to transit to a state one step closer to its goal, the desired state.

D. Timing

Timing is an important aspect of key rollovers, ignoring this could lead to an invalid DNSSEC situation. From an administrators point of view it is important to foresee when data becomes available to validators and when its recollection will be lost. Those timing parameters not only cover the TTLs, but also the delays introduced by software, zone transfers, and registration services.

E. Formalize DNSSEC Validity

The sections above describe that each record is in a state and has a desired state according to the associated key goal. The transition is only possible if the new state represents a valid DNSSEC situation. In other words, the transition must not break the chain of trust. In order to automatically verify this requirement a set of formal rules are defined that check the validity of a zone. The state machines together with the formal definition form a timed automata [10]. On the edges of the automata will be guards that represent the formal definition of DNSSEC validity. On the states will be timing guards that represent the various TTLs and delays. Thanks to this timed automata and the concept of key goals, a rollover can now easily be defined as putting goals on a set of keys.

F. Policy

There are several considerations that can influence the rollover strategy. For example, the so-called ZSK *Pre-Publish* rollover method [5] minimizes the size of the zone and responses during the process. The KSK *Double-Signature* approach [5] limits the number of parent interactions required. Local policy may enforce a particular strategy. However, the DNSSEC validity rules do not care about these considerations. Some special rules have to be added to differentiate between policy enforced rollover strategies.

IV. A New Way To Rollover

The current approach of replacing keys in DNSSEC is a non-trivial operation. Having defined our automata and the concept of key goals, we can specify our approach of rolling keys. Transitions are guarded by validity, time, and policy.

Algorithm 1 shows how the state changes are made to all keys of a zone in a single time-step. After a state change of a record, all other records must be re-evaluated since the record states are interdependent. After the run has completed, no more transitions can be made at this time. Because the timing information is available, an exact time can be determined when a next transition is possible. There is no need (nor harm) to run the algorithm again before that time.

It is possible that the zone reaches a complete stable state in which case passing of time does not trigger any state changes. In practice this means the zone needs not to be evaluated until some external event, such as a user requesting a new rollover.

Algorithm 1 Within a single time step, bring records closer to their goal. Return the nearest absolute time when useful work can be done.

```
nextRun ← ∞
repeat
  change ← ⊥
  for all key ∈ zone do
    for all record ∈ key do
      nextState ← desiredState(record_state, key_goal)
      if nextState = record_state then
        {This record is in a stable state}
        continue
      end if
      if not policyApproval(keyring, key, record, nextState) then
        {Local policy prevents transition}
        continue
      end if
      if not transitionAllowed(keyring, key, record, nextState)
      then
        {This transition would make the zone invalid}
        continue
      end if
      t ← transitionTime(record, nextState)
      if t > now() then
        {We are not allowed to make the transition at this time}
        nextRun ← minimum(t, nextRun)
        continue
      end if
      record_state ← nextState
      record_lastChange ← now()
      change ← ⊤
    end for
  end for
until not change
return nextRun
```

The algorithm introduces four functions:

desiredState(state, goal)
: Returns the "next" state in the state machine, given the goal and current state.

policyApproval(keyring, key, record, nextState)
: Evaluates if there are any local policies preventing the transition of this record to the next state. This function achieves that different flavours of rollovers are possible.

transitionAllowed(keyring, key, record, nextState)
: Evaluates the transition on correctness with respect to DNSSEC validity. See Section IV-A for more details.

transitionTime(record, nextState)
: Given a record and its desired next state, calculates

the point in time when the record may take the transition. This depends on the time of the previous transition for this record, the TTL for this type of record and additional introduced delays.

A. Formal Rules

In order to reason about the validity of a zone given an arbitrary set of keys in arbitrary state, we need a formal definition of that validity. Equations (3a) to (3i) give such a formal definition. This set of rules is evaluated with every attempt to transition a record from one state to another. If these rules hold, the transition can be carried out.

This system would never bring itself in a state where its rules no longer hold. However, it is imaginable that some external event brings the system in such a state and no single transition can bring it in a correct state. In this case, the rollback process should not deadlock or give up. To handle this situation it is allowed to remain in a bad state, but explicitly not allowed to get in to one. More specifically, a transition is allowed from key to key' if the proposition in Equation (1) holds (with $rule1$, $rule2$, and $rule3$ defined by Equation (3)).

$$\begin{aligned} &(\neg rule1(key) \vee rule1(key')) \quad \wedge \\ &(\neg rule2(key) \vee rule2(key')) \quad \wedge \\ &(\neg rule3(key) \vee rule3(key')) \end{aligned} \tag{1}$$

Each of the rules is a proposition about the state of the keys currently involved in the zone. Let x, y, z be keys. D, K, R, and S refer respectively to the resource records DS, DNSKEY, RRSIG DNSKEY, and RRSIG. The subscript of the record symbols D, K, R, and S denotes the key it belongs to, and the superscript the state it is in. States correspond to the state diagram from Figure 1: Hidden ($-$), Rumoured ($\uparrow$), Omnipresent ($+$), and Unretentive ($\downarrow$). For example $D_x^{\uparrow}$ indicates that the DS of key x is introduced in the zone, but might not be propagated to all caches. For brevity we allow multiple states as superscript: $D_x^{\uparrow+} \equiv D_x^{\uparrow} \vee D_x^{+}$. The equality sign ($=$) is used to indicate that resource records are in the same state.

We also define a recursive successor relation, Equation (2), $z \overset{T}{\succ} x$ where z is the successor of x for record type T.

$$\begin{aligned} z \overset{T}{\succ} x : Dep(x,T) = \emptyset \wedge \big(x \in Dep(z,T) \vee \\ \exists y \in Dep(z,T)\ (y \neq z \wedge y \overset{T}{\succ} x \wedge \\ D_y K_y R_y S_y = D_z K_z R_z S_z)\big) \end{aligned} \tag{2}$$

This relation refers to types of rollovers in which a certain record type is going to be *swapped*. For example, with the ZSK Pre-Publish rollover method the signatures created by the successor key z are being propagated first, so that the DNSKEY records for x and z can be swapped later on. In this case, we say that z is the successor of x for the DNSKEY record type.

Here, x is the predecessor key that is going to be withdrawn from the zone. The set $Dep(x,T)$ is a separately administrated set of keys that have a dependency on x for record type T. For example, with the ZSK Pre-Publish method, the DNSKEY of key x can be withdrawn if there is a succeeding DNSKEY of key z introduced in the zone. Key x now depends on key z, therefore x will be in the set $Dep(z,T)$. The successor relation requires that the predecessor key must not have any other keys relying on it. In other words, the set $Dep(x,T)$ must be empty.

It is possible to roll keys faster than the time required to finish the rollover procedure. For example, consider the keys x, y, z. Key x is currently published and is going to be replaced by y. The DNSKEY for x is removed from the zone and at the same moment the DNSKEY for y is introduced. Key y is a direct dependency for key x and is therefore the successor of x. However, before the new DNSKEY has been propagated, key z will replace key y. The DNSKEY for y is removed and moves into the same state as key x. Key y now directly depends on key z, and key z will be a new successor key for x. As long as the DNSKEY for z is not known to all caches, the RRSIGs of x, y, and z must actively be published.

Finally, let us define K as the complete set of keys associated with the zone, and X a subset of K with elements of the same cryptographic algorithm as key x: $X = \{k \in K | alg(k) = alg(x)\}$. The rules defining the validity of a zone are now given by:

$rule1(x)$:

$$\exists y \in K\ (D_y^{\uparrow+}) \tag{3a}$$

$rule2(x)$:

$$\exists y \in X\ (D_y^{+} K_y^{+} R_y^{+}) \quad \vee \tag{3b}$$
$$\exists y, z \in X\ (D_y^{\uparrow} K_y^{+} R_y^{+} D_z^{\downarrow} K_z^{+} R_z^{+} \wedge y \overset{D}{\succ} z) \quad \vee \tag{3c}$$
$$\exists y, z \in X\ (D_y^{+} K_y^{\uparrow+} R_y^{\uparrow} D_z^{+} K_z^{\downarrow} R_z^{\downarrow-} \wedge y \overset{K}{\succ} z) \quad \vee \tag{3d}$$
$$\forall y \in X\ (D_y^{-} \vee \exists z \in X\ (K_z^{+} R_z^{+} (D_y = D_z))) \tag{3e}$$

$rule3(x)$:

$$\exists y \in X\ (K_y^{+} S_y^{+}) \quad \vee \tag{3f}$$
$$\exists y, z \in X\ (K_y^{\uparrow} S_y^{+} K_z^{\downarrow} S_z^{+} \wedge y \overset{K}{\succ} z) \quad \vee \tag{3g}$$
$$\exists y, z \in X\ (K_y^{+} S_y^{\uparrow} K_z^{+} S_z^{\downarrow} \wedge y \overset{S}{\succ} z) \quad \vee \tag{3h}$$
$$\forall y \in X\ (K_y^{-} \vee \exists z \in X\ (S_z^{+} (K_y = K_z))) \tag{3i}$$

Each rule provides certainty about the validity of a *fraction* of the chain of trust. Rule 1 gives the requirements for the DS records, while Rule 2 does the same for the DNSKEY set. Rule 3 states the requirements for all other signatures. This differentiation allows recovery from an invalid state in a much more graceful manner, for example by making the distinction between a problem with the DNSKEY or the signatures.

It could be striking that all three propositions are functions of the key x, but the state of key x is never explicitly evaluated. In general the state of a key does not have influence on the validity of a zone as a whole, as long as there is some other

key that can be used to construct a chain of trust. Of course, key x can be the same key as y or z. Keys y and z may also be the same key.

Rule 1 in Equation (3a) simply states that there must be a DS record published at all times. This needs not be the DS record of key x, nor have the same cryptographic algorithm as x. This seems like an incomplete definition but is very important: Rule 2 does not require a published DS record to evaluate to true, but the state of the DS is of great influence. Rule 1 is the guard for having a signed zone. If Rule 1 is not enforced, the zone could roll to an unsigned situation or to another key *via* an unsigned situation.

Rule 2 ensures the DNSKEY set is in a correct state. Only keys in the propositions are considered with an algorithm equal to key x. Equation (3b) is the trivial case, there is a key with an omnipresent DS record and an omnipresent DNSKEY record (which is signed).

It is also possible, see Equation (3c), that there are two or more keys with an omnipresent DNSKEY and the DS records get swapped. These keys must be in a successor relation, either direct or indirect. A validator is then guaranteed to have at least one of the DS records available in order to be able to construct the chain of trust. Similarly, the DNSKEYs can be swapped if the DS records are omnipresent, Equation (3d).

Equation (3e) is somewhat more complicated and deals with the unsigned situation. It should be read as: key x may be in any state as long as all other keys y have their DS hidden or, when their DS is not hidden, there must be a key z with its DS in the same state and its DNSKEY omnipresent. Of course y and z can be the same key. In other words, if a DS record for this cryptographic algorithm is still available to some validators, there *must* be a chain of trust *for those validators*. With Equation (3e), multiple DS records can be withdrawn at the same time.

Rule 3 is very similar to Rule 2 but reasons about the signatures. Again only keys with the same algorithm as x are considered. At all times, one of the Equations (3f)–(3i) should hold. Equation (3f): There is both a DNSKEY and signatures of one key known to all validators. Equation (3g): All validators have access to at least one of the DNSKEYs of keys y to z and all the signatures. A chain of trust can be build either way. Equation (3h): All validators have access to one of the signatures of keys y to z and all the DNSKEYs. Equation (3i): If no DNSKEYs are published, the state of the signatures is irrelevant. In case a DNSKEY is published however, there must be a path that can be validated from there.

B. Enforcing Policy

When applying this algorithm on a set of keys, the state of the records will transit to the key goals as fast as the validity and timing constraints would allow them. However, the shortest path is not always desirable. An operator may choose to trade some speed with bandwidth or the number of administrative transactions. So far, we have identified three (partially) conflicting policy strategies:

Minimize the amount of parent interactions
: Introducing a new DS record at the parent and removing an old at the same time. In order to ensure a chain of trust, the other records of the new key's DNSKEY must be omnipresent before doing this.

Minimize the size of the DNSKEY set
: As above, but swap the old DNSKEY with the new DNSKEY. Depending on the type of key, the DS record and/or the RRSIG records must be in omnipresent state.

Minimize signatures
: Signing all data with multiple keys will increase bandwidth usage significantly. Signatures can be swapped in an atomic operation as long as the DNSKEYs are omnipresent.

To achieve one of these strategies one must effectively hold back transitions until some requirement is fulfilled. These requirements are not related to the validity of the zone, the formal rules would allow the transition. Thus policy is considered to be extra barriers in the system. In Algorithm 1, this is handled by the *policyApproval* function. This distinction allows us to add future policy demands, without having to interfere with the timing parameters or the rules on DNSSEC validation.

V. Implementation

A. OpenDNSSEC

The model and rules proposed in this paper are developed as part of the OpenDNSSEC project. OpenDNSSEC is a set of applications that automate DNSSEC signing, key management, and distribution of the signed zone. The component responsible for key management is called the *enforcer*. As DNSSEC deployment is still quite recent, new insights and requirements were added during OpenDNSSEC development, and some of these additions were hard to incorporate in the initial design. This led to the development of a new enforcer and the research described here. At the time of writing, the software is functional and feature complete but not rigorously tested and therefore still in alpha stage.

Our implementation stores the policy parameters with which a key is initially created with the key it self. In effect we can, next to arbitrary key rollovers, also change policy and timing parameters without any special treatment.

B. Example of ZSK Double Signature Rollover

The examples in this paper are output traces of our current implementation. We will take a closer look at one of them, other examples can be found in the appendix for reference. Names for the rollover scenarios are taken from "DNSSEC Key Timing Considerations" [9].

In the example at hand, Table I presents the steps in the rollover process between two keys. The columns represent records which are grouped by key. Each row is a time step. When a record does not change during a step, the record's corresponding cell is left blank. Unless stated otherwise, all keys are of the same cryptographic algorithm.

Without any policy directives, a ZSK Double Signature rollover will be executed, as this gives the shortest path to a stable situation.

KSK1 (in)	ZSK1 (out)	ZSK2 (in)	Time
$D^+ \quad K^+ \quad R^+$	$K^+ \quad S^+$	$K^- \quad S^-$	
		$K^\uparrow \quad S^\uparrow$	0
	$K^\downarrow \quad S^\downarrow$	$K^+ \quad S^+$	$MaxTTL(key, sig)$
	$K^- \quad S^-$		$MinTTL(key, sig)$
Total time: $TTL(sig) + TTL(key)$			

TABLE I
ZSK DOUBLE SIGNATURE ROLLOVER

Let us walk through the example in Table I. We will evaluate each record from left to right. The KSK is ignored in our description as there are no changes during the rollover. In this demonstration, we will assume that the TTLs for all records are the same.

Since ZSK1 is being replaced, the rollover process tries to change the records of ZKS1 to the hidden state ($-$). For the DNSKEY (K), this means next state will be unretentive ($\downarrow$). Currently all three rules evaluate to true. But if the change is applied, Rule 3 can no longer be satisfied. Specifically $\exists y \in X\ (K_y^+ S_y^+)$ would no longer hold. The signatures (S) of this key fail to transition for the same reason.

The three rules are also true for the DNSKEY and RRSIG for ZSK2. ZSK1 and KSK1 form a chain of trust for this cryptographic algorithm; no change to ZSK2 could break that chain. After transition of both records, the rules will still hold, hence both are moved to the next state. Tied to this change of course, is the actual publication of the records.

After this step, an iteration is made over all records again. But ZSK1 is still blocked for the same reason. ZSK2 is blocked by TTL.

The second time step (row 3) will happen as soon as the records of ZSK2 have been in rumoured state long enough. These records may transition to omnipresent for the same reason they could transition to rumoured before. After this (in the same time step), a chain of trust can be built with KSK1 and ZSK2. Changing the state of the records of ZSK1 does not influence the validity of the zone any longer, thus both records may transition to unretentive. From now on these records are no longer published.

In the last step, after sufficient time has passed, both records of ZSK1 may enter the hidden state as no validator has this data in cache anymore.

VI. SUMMARY

Key rollovers in DNSSEC are a complex and error prone process. In this paper, we have presented a new, cache centric approach to key rollover that has a number of advantages over existing common approaches. The new method is flexible and supports CSK and algorithm rollover, making them no more exceptional than regular KSK and ZSK rollovers. The ability to switch between ongoing rollovers makes emergency rollovers very efficient. The method is also robust, in that it can recover from an invalid state due to an external event.

The concepts and approach presented in this paper are also relevant to other implementations, existing or future. These implementations do not necessarily include the complete mechanism of interdependent state machines and rollover process, but can also only incorporate the formal rules as a correctness check in their procedural approach.

We have designed and implemented a prototype of the cache centric approach to key rollover. The prototype is functional and feature complete, but still in alpha stage release. Integration of the prototype with OpenDNSSEC is planned for the near future.

Other future work is support for key revocation [11] to provide automated updates of trust anchors. Also, the concept of standby keys [9] is not incorporated. To assert the correctness of the formal rules defined in Section IV-A, a formal model checking model can be used to validate the rules set.

ACKNOWLEDGMENTS

The authors would like to thank Wouter Wijngaards for his valuable comments.

REFERENCES

[1] P. Mockapetris and J. Dunlap, "Development of the domain name system," in *Proceeding of the 1988 Symposium on Communications Architectures and Protocols (SIGCOMM '88)*, Stanford, CA, Aug. 1988, pp. 123–133.

[2] S. M. Bellovin, "Using the domain name system for system break-ins," in *Proceedings of the Fifth Usenix Unix Security Symposium*, Salt Lake City, UT, Jun. 1995, pp. 199–208.

[3] D. Schneider, "Fresh phish," *IEEE Spectrum*, vol. 45, no. 10, pp. 34–38, Oct. 2008.

[4] A. Friedlander, A. Mankin, W. D. Maughan, and S. D. Crocker, "DNSSEC: A protocol toward securing the Internet infrastructure," *Communications of the ACM*, vol. 50, no. 6, pp. 44–50, Jun. 2007.

[5] O. Kolkman and R. Gieben, "DNSSEC operational practices," RFC Editor, RFC 4641, Sep. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4641.txt

[6] S. Krishnaswamy, W. Hardaker, and R. Mundy, "DNSSEC in practice: Using DNSSEC-tools to deploy DNSSEC," in *Proceedings of the Cybersecurity Applications & Technology Conference for Homeland Security (CATCH'09)*, Washington, DC, Mar. 2009, pp. 3–15.

[7] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, "Deploying cryptography in Internet-scale systems: A case study on DNSSEC," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 656–669, Sep./Oct. 2011.

[8] M. Santcroos and O. Kolkman, "DNS threat analysis," NLnet Labs, Technical Report 2006-SE-01, May 2007.

[9] S. Morris, J. Ihren, and J. Dickinson, "DNSSEC key timing considerations," Work in progress, IETF Secretariat, Internet-Draft draft-ietf-dnsop-dnssec-key-timing, 2011. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-dnsop-dnssec-key-timing/

[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[11] M. StJohns, "Automated updates of DNS security (DNSSEC) trust anchors," RFC Editor, RFC 5011, Sep. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5011.txt

APPENDIX
ADDITIONAL EXAMPLES OF KEY ROLLOVERS

The examples are presented in tables and describe the rollover between keys. The columns represent records which are grouped by key. Each row is a time step. When a record does not change during a step, the record's corresponding cell is left blank. Unless stated otherwise, all keys are of the same cryptographic algorithm.

In a ZSK Pre-Publication rollover scenario, Table II, the DNSKEY is introduced while the signatures are held back. Once the DNSKEY is Omnipresent, the signatures can be switched. For a while, multiple DNSKEY records are published.

KSK1 (in)			ZSK1 (out)		ZSK2 (in)		Time
D^+	K^+	R^+	K^+	S^+	K^-	S^-	
					$K^{\uparrow}$		0
				$S^{\downarrow}$	K^+	$S^{\uparrow}$	$TTL(key)$
			$K^{\downarrow}$	S^-		S^+	$TTL(sig)$
			K^-				$TTL(key)$
Total time: $2 \times TTL(key) + TTL(sig)$							

TABLE II
ZSK PRE-PUBLICATION ROLLOVER

The ZSK Double RRSIG rollover in Table III is similar, but introduces the signatures first and then switches DNSKEY records. Typically, the TTL on keys is longer than the TTL of the zone data, making this rollover somewhat quicker then the previous. As a drawback, every DNS response contains the signatures of both keys for a certain period.

KSK1 (in)			ZSK1 (out)		ZSK2 (in)		Time
D^+	K^+	R^+	K^+	S^+	K^-	S^-	
						$S^{\uparrow}$	0
			$K^{\downarrow}$		$K^{\uparrow}$	S^+	$TTL(sig)$
			K^-	$S^{\downarrow}$	K^+		$TTL(key)$
				S^-			$TTL(sig)$
Total time: $2 \times TTL(sig) + TTL(key)$							

TABLE III
ZSK DOUBLE RRSIG ROLLOVER

The KSK Double DS rollover in Table IV is very similar to the ZSK rollovers we showed. First the new DS record is introduced so the DNSKEYs can be swapped.

KSK1 (out)			KSK2 (in)			ZSK1 (in)		Time
D^+	K^+	R^+	D^-	K^-	R^-	K^+	S^+	
			$D^{\uparrow}$					0
	$K^{\downarrow}$	$R^{\downarrow}$	D^+	$K^{\uparrow}$	$R^{\uparrow}$			$TTL(ds)$
$D^{\downarrow}$	K^-	R^-		K^+	R^+			$TTL(key)$
D^-								$TTL(ds)$
Total time: $TTL(key) + 2 \times TTL(ds)$								

TABLE IV
KSK DOUBLE DS ROLLOVER

Table V, KSK Double Signature rollover, has the same effect but now the DS records are swapped. For changing the DS record, interaction with the parent is needed. This rollover is especially useful if this is a slow or cumbersome process, as it needs only one of such interactions.

KSK1 (out)			KSK2 (in)			ZSK1 (in)		Time
D^+	K^+	R^+	D^-	K^-	R^-	K^+	S^+	
				$K^{\uparrow}$	$R^{\uparrow}$			0
$D^{\downarrow}$			$D^{\uparrow}$	K^+	R^+			$TTL(key)$
D^-	$K^{\downarrow}$	$R^{\downarrow}$	D^+					$TTL(ds)$
	K^-	R^-						$TTL(key)$
Total time: $2 \times TTL(key) + TTL(ds)$								

TABLE V
KSK DOUBLE SIGNATURE ROLLOVER

Our last example, Table VI, shows how the key replacement process works if we want to rollover to a different cryptographic algorithm and go from a KSK-ZSK split towards a single type (CSK) signing scheme. Because the keys are using a different cryptographic algorithm, the order of withdrawing and introducing records is much more strict. Only when the new key provides a full chain of trust, the old one is allowed to be withdrawn.

KSK1 (out)			ZSK1 (out)		CSK1 (in)				Time
D^+	K^+	R^+	K^+	S^+	D^-	K^-	R^-	S^-	
								$S^{\uparrow}$	0
						$K^{\uparrow}$	$R^{\uparrow}$	S^+	$TTL(sig)$
$D^{\downarrow}$					$D^{\uparrow}$	K^+	R^+		$TTL(key)$
D^-	$K^{\downarrow}$	$R^{\downarrow}$	$K^{\downarrow}$		D^+				$TTL(ds)$
	K^-	R^-	K^-	$S^{\downarrow}$					$TTL(key)$
				S^-					$TTL(sig)$
Total time: $2 \times TTL(key) + 2 \times TTL(sig) + TTL(ds)$									

TABLE VI
KSK-ZSK SPLIT TO CSK SIGNING SCHEME ALGORITHM ROLLOVER