Internet Draft
draft-ihren-dnsext-threshold-validation-01.txt
July 2004
Expires in six months

Johan Ihren
Autonomica
Bill Manning
EP.NET

Threshold Validation:

A Mechanism for Improved Trust and Redundancy for DNSSEC Keys

Status of this Memo

Abstract

This memo documents a proposal for a different method of validation for DNSSEC aware resolvers. The key change is that by changing from a model of one Key Signing Key, KSK, at a time to multiple KSKs it will be possible to increase the aggregated trust in the signed keys by leveraging from the trust associated with the different signees.

By having multiple keys to chose from validating resolvers get the opportunity to use local policy to reflect actual trust in different keys. For instance, it is possible to trust a single, particular key ultimately, while requiring multiple valid signatures by less trusted keys for validation to succeed. Furthermore, with multiple KSKs there are additional redundancy benefits available since it is possible to roll over different KSKs at different times which may make rollover scenarios easier to manage.

Contents

1. Terminology

The key words "MUST", "SHALL", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119.

The term "zone" refers to the unit of administrative control in the Domain Name System. "Name server" denotes a DNS name server that is authoritative (i.e. knows all there is to know) for a DNS zone, typically the root zone. A "resolver", is a DNS "client", i.e. an entity that sends DNS queries to authoritative nameservers and interpret the results. A "validating resolver" is a resolver that attempts to perform DNSSEC validation on data it retrieves by doing DNS lookups.

2. Introduction and Background

From a protocol perspective there is no real difference between different keys in DNSSEC. They are all just keys. However, in actual use there is lots of difference. First and foremost, most DNSSEC keys have in-band verification. I.e. the keys are signed by some other key, and this other key is in its turn also signed by yet another key. This way a "chain of trust" is created. Such chains have to end in what is referred to as a "trusted key" for validation of DNS lookups to be possible.

A "trusted key" is a the public part of a key that the resolver acquired by some other means than by looking it up in DNS. The trusted key has to be explicitly configured in the resolver.

A node in the DNS hierarchy that issues such out-of-band "trusted keys" is called a "security apex" and the trusted key for that apex is the ultimate source of trust for all DNS lookups within that entire subtree.

DNSSEC is designed to be able to work with more than one security apex. These apexes will all share the problem of how to distribute their "trusted keys" in a way that provides validating resolvers with confidence in the distributed keys.

Maximizing that confidence is crucial to the usefulness of DNSSEC and this document tries to address this issue.

3. Trust in DNSSEC Keys

In the end the trust that a validating resolver will be able to put in a key that it cannot validate within DNSSEC will have to be a function of

- trust in the key issuer, aka the KSK holder

- trust in the distribution method

- trust in extra, out-of-band verification

A KSK holder needs to be trusted not to accidentally lose private keys in public places. Furthermore it needs to be trusted to perform correct identification of the ZSK holders in case they are separate from the KSK holder itself.

The distribution mechanism can be more or less tamper-proof. If the key holder publishes the public key, or perhaps just a secure fingerprint of the key in a major newspaper it may be rather difficult to tamper with. A key acquired that way may be easier to trust than if it had just been downloaded from a web page.

Out-of-band verification can for instance be the key being signed by a certificate issued by a known Certificate Authority that the resolver has reason to trust.

3.1. Simplicity vs Trust

The fewer keys that are in use the simpler the key management becomes. Therefore increasing the number of keys should only be considered when the complexity is not the major concern. A perfect example of this is the distinction between so called Key Signing Keys, KSK, and Zone Signing Keys, ZSK. This distinction adds overall complexity but simplifies real life operations and was an overall gain since operational simplification was considered to be a more crucial issue than the added complexity.

In the case of a security apex there are additional issues to consider, among them

* maximizing trust in the KSK received out-of-band

* authenticating the legitimacy of the ZSKs used

In some cases this will be easy, since the same entity will manage both ZSKs and KSKs (i.e. it will authenticate itself, somewhat similar to a self-signed certificate). In some environments it will be possible to get the trusted key installed in the resolver end by decree (this would seem to be a likely method within corporate and government environments).

In other cases, however, this will possibly not be sufficient. In the case of the root zone this is obvious, but there may well be other cases.

3.2. Expanding the "Trust Base"

For a security apex where the ZSKs and KSK are not held by the same entity the KSK will effectively authenticate the identity of whoever does real operational zone signing. The amount of trust that the data signed by a ZSK will get is directly dependent on whether the end resolver trusts the KSK or not, since the resolver has no OOB access to the public part of the ZSKs (for practical reasons).

Since the KSK holder is distinct from the ZSK holder the obvious question is whether it would then be possible to further improve the situation by using multiple KSK holders and thereby expanding the trust base to the union of that available to each individual KSK holder. "Trust base" is an invented term intended to signify the aggregate of Internet resolvers that will eventually choose to trust a key issued by a particular KSK holder.

A crucial issue when considering trust expansion through addition of multiple KSK holders is that the KSK holders are only used to authenticate the ZSKs used for signing the zone. I.e. the function performed by the KSK is basically:

"This is indeed the official ZSK holder for this zone,
I've verified this fact to the best of my abilitites."

Which can be thought of as similar to the service of a public notary. I.e. the point with adding more KSK holders is to improve the public trust in data signed by the ZSK holders by improving the strength of available authentication.

Therefore adding more KSK holders, each with their own trust base, is by definition a good thing. More authentication is not controversial. On the contrary, when it comes to authentication, the more the merrier.

4. Proposed Semantics for Signing the KEY Resource Record Set

In DNSSEC according to RFC2535 all KEY Resource Records are used to sign all authoritative data in the zone, including the KEY RRset itself, since RFC2535 makes no distinction between Key Signing Keys, KSK, and Zone Signing Keys, ZSK. With Delegation Signer [DS] it is possible to change this to the KEY RRset being signed with all KSKs and ZSKs but the rest of the zone only being signed by the ZSKs.

This proposal changes this one step further, by recommending that the KEY RRset is only signed by the Key Signing Keys, KSK, and explicitly not by the Zone Signing Keys, ZSK. The reason for this is to maximize the amount of space in the DNS response packet that is available for additional KSKs and signatures thereof. The rest of the authoritative zone contents are as previously signed by only the ZSKs.

4.1. Packet Size Considerations

The reason for the change is to keep down the size of the aggregate of KEY RRset plus SIG(KEY) that resolvers will need to acquire to perform validation of data below a security apex. For DNSSEC data to be returned the DNSSEC OK bit in the EDNS0 OPT Record has to be set, and therefore the allowed packet size can be assumed to be at least the EDNS0 minimum of 4000 bytes.

When querying for KEY + SIG(KEY) for "." (the case that is assumed to be most crucial) the size of the response packet after the change to only sign the KEY RR with the KSKs break down into a rather large space of possibilities. Here are a few examples for the possible alternatives for different numbers of KSKs and ZSKs for some different key lengths (all RSA keys, with a public exponent that is < 254). This is all based upon the size of the response for the particular example of querying for

". KEY IN"

with a response of entire KEY + SIG(KEY) with the authority and additional sections empty:

```
ZSK/768 and KSK/1024   (real small)
Max 12 KSK +  3 ZSK  at 3975
    10 KSK +  8 ZSK  at 3934
     8 KSK + 13 ZSK  at 3893

ZSK/768 + KSK/1280
MAX 10 KSK +  2 ZSK  at 3913
     8 KSK +  9 ZSK  at 3970
     6 KSK + 15 ZSK  at 3914

ZSK/768 + KSK/1536
MAX  8 KSK +  4 ZSK  at 3917
     7 KSK +  8 ZSK  at 3938
     6 KSK + 12 ZSK  at 3959

ZSK/768 + KSK/2048
MAX  6 KSK +  5 ZSK  at 3936
     5 KSK + 10 ZSK  at 3942

ZSK/1024 + KSK/1024
MAX 12 KSK +  2 ZSK  at 3943
    11 KSK +  4 ZSK  at 3930
    10 KSK +  6 ZSK  at 3917
     8 KSK + 10 ZSK  at 3891

ZSK/1024 + KSK/1536
MAX  8 KSK +  3 ZSK  at 3900
     7 KSK +  6 ZSK  at 3904
     6 KSK +  9 ZSK  at 3908

ZSK/1024 + KSK/2048
MAX  6 KSK +  4 ZSK  at 3951
     5 KSK +  8 ZSK  at 3972
     4 KSK + 12 ZSK  at 3993
```

Note that these are just examples and this document is not making any recommendations on suitable choices of either key lengths nor number of different keys employed at a security apex.

This document does however, based upon the above figures, make the recommendation that at a security apex that expects to distribute "trusted keys" the KEY RRset should only be signed with the KSKs and not with the ZSKs to keep the size of the response packets down.

5. Proposed Use of Multiple "Trusted Keys" in a Validating Resolver

In DNSSEC according to RFC2535[RFC2535] validation is the process of tracing a chain of signatures (and keys) upwards through the DNS hierarchy until a "trusted key" is reached. If there is a known trusted key present at a security apex above the starting point validation becomes an exercise with a binary outcome: either the validation succeeds or it fails. No intermediate states are possible.

With multiple "trusted keys" (i.e. the KEY RRset for the security apex signed by multiple KSKs) this changes into a more complicated space of alternatives. From the perspective of complexity that may be regarded as a change for the worse. However, from a perspective of maximizing available trust the multiple KSKs add value to the system.

5.1. Possible to do Threshold Validation

With multiple KSKs a new option that opens for the security concious resolver is to not trust a key individually. Instead the resolver may decide to require the validated signatures to exceed a threshold. For instance, given M trusted keys it is possible for the resolver to require N-of-M signatures to treat the data as validated.

I.e. with the following pseudo-configuration in a validating resolver

```
security-apex "." IN {
   keys { ksk-1 .... ;
          ksk-2 .... ;
          ksk-3 .... ;
          ksk-4 .... ;
          ksk-5 .... ;
        };
   validation {
      # Note that ksk-4 is not present below
      keys { ksk-1; ksk-2; ksk-3; ksk-5; };
      # 3 signatures needed with 4 possible keys, aka 75%
      needed-signatures 3;
   };
};
```

we configure five trusted keys for the root zone, but require three valid signatures for the top-most KEY for validation to succeed. I.e. threshold validation does not force multiple signatures on the entire signature chain, only on the top-most signature, closest to the security apex for which the resolver has trusted keys.

5.2. Not All Trusted Keys Will Be Available

With multiple KSKs held and managed by separate entities the end resolvers will not always manage to get access to all possible trusted keys. In the case of just a single KSK this would be fatal to validation and necessary to avoid at whatever cost. But with several fully trusted keys available the resolver can decide to trust several of them individually. An example based upon more pseudo-configuration:

```
     security-apex "." IN {
        keys { ksk-1 .... ;
               ksk-2 .... ;
               ksk-3 .... ;
               ksk-4 .... ;
               ksk-5 .... ;
             };
        validation {
           # Only these two keys are trusted independently
           keys { ksk-1; ksk-4; };
           # With these keys a single signature is sufficient
           needed-signatures 1;
        };
     };
```

Here we have the same five keys and instruct the validating resolver to fully trust data that ends up with just one signature from by a fully trusted key.

The typical case where this will be useful is for the case where there is a risk of the resolver not catching a rollover event by one of the KSKs. By doing rollovers of different KSKs with different schedules it is possible for a resolver to "survive" missing a rollover without validation breaking. This improves overall robustness from a management point of view.

5.3. Not All Possible KSKs Need to Be Trusted

With just one key available it simply has to be trusted, since that is the only option available. With multiple KSKs the validating resolver immediately get the option of implementing a local policy of only trusting some of the possible keys.

This local policy can be implemented either by simply not configuring keys that are not trusted or, possibly, configure them but specify to the resolver that certain keys are not to be ultimately trusted alone.

6. Additional Benefits from Having Multiple KSKs

6.1. More Robust Key Rollovers

With only one KSK the rollover operation will be a delicate operation since the new trusted key needs to reach every validating resolver before the old key is retired. For this reason it is expected that long periods of overlap will be needed.

With multiple KSKs this changes into a system where different "series" of KSKs can have different rollover schedules, thereby changing from one "big" rollover to several "smaller" rollovers.

If the resolver trusts several of the available keys individually then even a failure to track a certain rollover operation within the overlap period will not be fatal to validation since the other available trusted keys will be sufficient.

6.2. Evaluation of Multiple Key Distribution Mechanisms

Distribution of the trusted keys for the DNS root zone is recognized to be a difficult problem that ...

With only one trusted key, from one single "source" to distribute it will be difficult to evaluate what distribution mechanism works best. With multiple KSKs, held by separate entitites it will be possible to measure how large fraction of the resolver population that is trusting what subsets of KSKs.

KSK holders SHOULD therefore try to use methods as diverse as possible to publish the public part of the KSK.

7. Security Considerations

From a systems perspective the simplest design is arguably the best, i.e. one single holder of both KSK and ZSKs. However, if that is not possible in all cases a more complex scheme is needed where additional trust is injected by using multiple KSK holders, each contributing trust, then there are only two alternatives available. The first is so called "split keys", where a single key is split up among KSK holders, each contributing trust. The second is the multiple KSK design outlined in this proposal.

Both these alternatives provide for threshold mechanisms. However split keys makes the threshold integral to the key generating mechanism (i.e. it will be a property of the keys how many signatures are needed). In the case of multiple KSKs the threshold validation is not a property of the keys but rather local policy in the validating resolver. A benefit from this is that it is possible for different resolvers to use different trust policies. Some may configure threshold validation requiring multiple signatures and specific keys (optimizing for security) while others may choose to accept a single signature from a larger set of keys (optimizing for redundancy). Since the security requirements are different it would seem to be a good idea to make this choice local policy rather than global policy.

Furthermore, a clear issue for validating resolvers will be how to ensure that they track all rollover events for keys they trust. Even with operlap during the rollover (which is clearly needed) there is still a need to be exceedingly careful not to miss any rollovers (or fail to acquire a new key) since without this single key validation will fail. With multiple KSKs this operation becomes more robust, since different KSKs may roll at different times according to different rollover schedules. Therefore losing one key, for whatever reason, will not be crucial unless the resolver intentionally chooses to be completely dependent on that exact key.

8. IANA Considerations.

NONE.

9. References

9.1. Normative.

[RFC2535] Domain Name System Security Extensions. D. Eastlake. March 1999.

[RFC3090] DNS Security Extension Clarification on Zone Status. E. Lewis. March 2001.

9.2. Informative.

[RFC3110] RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS). D. Eastlake 3rd. May 2001.

[RFC3225] Indicating Resolver Support of DNSSEC. D. Conrad. December 2001.

[DS] Delegation Signer Resource Record. O. Gudmundsson. October 2002. Work In Progress.

10. Acknowledgments.

We've had much appreciated help from (in no particular order) Jakob Schlyter, Paul Vixie, Olafur Gudmundson and Olaf Kolkman.

11. Authors' Addresses

Johan Ihren
Autonomica AB
Bellmansgatan 30
SE-118 47 Stockholm, Sweden
johani@autonomica.se

Bill Manning

EP.NET
Marina del Rey, CA, USA
bmanning@ep.net