Chain of Trust

The parent-child and keyholder-keysigner relations and their communication in DNSSEC

R. Gieben
Stichting NLnet Labs
miek@nlnetlabs.nl
http://www.nlnetlabs.nl

Abstract DNSSEC is the intended successor of DNS (Domain Name System), the well-known system that takes care of the mapping between domain names and IP numbers on the internet. There are however some security problems with DNS, thus the need for DNSSEC (DNS SECure). DNSSEC uses public key cryptography to solve the security issues in the DNS. The goal of DNSSEC is to create a chain of trust in which a top level zone (like com) signs the key of a lower zone (such as child.com) which in turn can sign a even lower zone (a.child.com, for instance). To set up this chain, keys must be exchanged and signatures must be renewed on a regular basis. Furher more, keys can be discoverd, lost or stolen. This master thesis delves into those problems and presents possible solutions and procedures for efficient and (reasonably) safe distribution and renewal of keys and signatures.
Keywords: Distributed databases, Security and protection, Applications, Public key cryptosystems.
Classification: 68P25, 68M10 (MSC 2000);
C.2.4, C.2.0, C.2.2, E.3 (CR'98).

Contents

1 Introduction

In the early days the Internet was a place where friends and scientists could talk to each other. It was used as a medium where they could exchange information and ideas. Security was not much of an issue. Today, with more and more people using the Internet for business, it is a much more hostile environment with email viruses, denial of service (DoS) attacks, crackers, script kiddies, spoofing, etc, etc. These threats aren't merely an annoyance anymore; they could cost a lot of money. It is already obvious that these trends are not going to stop. The cash flow and the hacking/cracking on the Net are only going to increase.

One of such systems on the Internet that works great in a friendly environment, but is vulnerable now, is the domain name system (DNS). The primary job of the DNS is to take care of the mapping between domain names (like www.example.com) and IP numbers (like 192.168.1.1). This system has proven to be highly scalable and the Internet would not have grown so big, so fast, without it. There are however also some security problems with DNS. It has been shown that it is possible to corrupt the system with false data. Instead of having www.example.com mapping to 192.168.1.1 an attacker could make it point to an arbitrary IP address under his control.

To address this and other shortcomings of the DNS an addition has been proposed, called the secure domain name system (DNSSEC). DNSSEC uses cryptographically signed responses to authenticate the results, thus detecting falsified data. Although DNSSEC solves the security issues in DNS it creates other problems:

CPU intensive DNSSEC uses public key encryption to sign data. This is very computational so the less data you sign, the better.

zone file sizes DNS stores its information in zone files. Some of those file are already very big. With DNSSEC these files may be six fold in size.

key management DNSSEC uses keys and signatures for authentication, and it is yet unknown how to handle those (millions) of keys.

administrative issues there is no experience (yet) how to handle the extra overhead DNSSEC is creating.

more involvement DNSSEC will need much more involvement from system administrators.

how to start DNS was rolled out when the Internet was small. Today there are millions of hosts that need to be converted to DNSSEC.

Most problems DNSSEC creates seem solvable, but the key management issue could be a big problem. There are trials done with DNSSEC today, but those haven't reached the point yet to have problems with key management. Either this issue is solved and DNSSEC is rolled out or it isn't and an alternative must be found. This master thesis provides a possible solution to the key management issue (see section 5) and also describes a number of procedures that TLD can follow when implementing DNSSEC. All these procedures use old, known keys to authenticate new requests from delegated zones, thus reducing the problem on how to get the first key authenticated.

This "first key issue" is solved using DNS in the most secure means possible and using the existing DNS infrastructure. When this key has been signed by a TLD it must be sent back to the child for inclusion in the zonefile. This extra step of sending the resulting signature back to the child is also something I challenge. A large TLD wants to reduce to need to communicate with its children. Inclusion of the signature in the parent's zone is a way of solving this. This is however still a very controversial thing to do and it is uncertain how the rest of the DNSSEC community will react to this idea.

If DNSSEC is rolled out it also provides an infrastructure to publish public keys or other certificates in a secure manner. Thus it will not only provide a secure DNS, but also a worldwide database with public keys people can use to encrypt, decrypt and authenticate digital communications. This is something a lot of people and companies are eagerly awaiting. DNSSEC is a major step to a more secure Internet.

The security problem with DNS was identified in 1990 [1]. After thinking about solutions for almost 5 years the RFC describing DNSSEC was released in 1995 [11]. It has taken another 5 years to complete the tools implementing

DNSSEC. Only now (2001) there are projects on the way to convert the DNS tree to a DNSSEC tree. One such a project is .nl.nl. The purpose of that project is to mirror the .nl (DNS)tree in a secure fashion. The project has revealed that there are still problems left. The DNSSEC specification states that public key encryption is used, but it does not specify how to manage those millions of keys, nor does it specify the problems arising when (private) keys get compromised or when child zones wish to use new keys for their zones.

This master thesis is a project for my work at NLnet Labs in order to graduate from the University of Nijmegen, the main goal of this project will be:

> **I.** Define a set of procedures that implement a safe and secure communication between the parent-child and keyholder-keysigner relation in DNSSEC.

Concretely I will look at the impact DNSSEC will have on large ccTLD's and gTLD's. The procedures needed for .nl.nl are very much the same as the ones in use now (getting a new domain, changing data concerning a domain, etc.), but there are a few important new ones, concerning the administration of public keys. A zone can now, for instance, decide they want to use a new keys. How do you handle such an update? Another big issue is authentication. How do you know that if somebody claims that a key belongs to a particular zone, this really is the case? In this master thesis such procedures are described and any additional problems are identified and possibly solved. The procedures are detailed in sections 5 and 7.

Another, more concrete, goal is to:

> **II.** Find a possibly solution for the key distribution problem and specifically where the parent generated signatures should be located.

Generally the parent signs the key of the child and the child stores this signature in its zonefile. The results found when doing this research challenge this, especially key management looks very difficult in such a setup. In section 7, paragraph 6.3.1, a possible solution is given for this problem. This proposal is currently an Internet draft [28].

Finally the following problems are looked at more closely.

- DNSSEC is much more CPU intensive than DNS. How to cope with this? (Described in section 7)
- DNSSEC zonefiles are 3-6 times larger than a DNS zone file. How do you handle this much data? (Section 7)
- there is not (yet) any experience with the extra overhead DNSSEC is creating. (Section 4)
- how to start with deployment of DNSSEC? (Section 8)

1.1 Recommended reading

The reader of this document is assumed to have some basic knowledge of the Domain Name System. The DNS & BIND book [2] is a good introduction and the article from Steven M. Belovin [1] clearly explains a major loophole in DNS. The following RFC's are also relevant to the topic: [6, 7] and [11]. There is also a heavy use of acronyms in this master thesis. In section 10 on page 43 the most common ones are listed and explained.

1.2 Examples used in this master thesis

In all the examples I will use the domain example.com and all information associated with it in the (actual) DNS, like IP numbers. All other addresses are local area addresses (LAN) as specified in [10].

2 The Domain Name System

Since the early days of the Internet there is a need for a system that allows computers to be found on a network. At that time there were only a few hundred hosts. A single file, *HOSTS.TXT*, contained all the information you needed to know about those hosts, like their IP addresses.

HOSTS.TXT was maintained by SRI's Network Information Center[1] (the NIC) and distributed from a single host: SRI-NIC. ARPAnet administrators emailed updates to this file to the NIC, and periodically *ftp*ed to SRI-NIC to get the latest HOSTS.TXT. When the Internet grew bigger and bigger it became apparent that this scheme would no longer work.

The main problems were:

traffic and load the SRI-NIC server couldn't handle much more in terms of processor load and network traffic.

name collisions no two hosts could have the same name. There was also no mechanism preventing from someone adding a host with a conflicting name and breaking the whole scheme.

consistency the Internet became so large that by the time everybody had the latest *HOSTS.TXT* file, there was already a new and updated version.

To solve these problems the Domain Name System (DNS) was invented. The goal of DNS is to create a distributed, hierarchical database and make each administrator responsible for their own little part of the database, called a zone or more popular a domain.

Paul Mockapetris was responsible for designing the architecture of the new system, documented in [6] and [7].

[1] Stanford Research Institute, see http://www.sri.com.

2.1 DNS design and structure

In short, DNS is a distributed database constructed in a tree like fashion. In this scheme there are servers (*nameservers*) and there are clients (*resolvers*). Figure 1 shows a part of the DNS tree. The root of the tree is denoted as a single dot ("."). Each node is also the root of a sub tree and is called a *domain*. A domain must have at least two authoritative nameservers. Although this is more an implementation issue then something directly specified in the relevant RFC's.

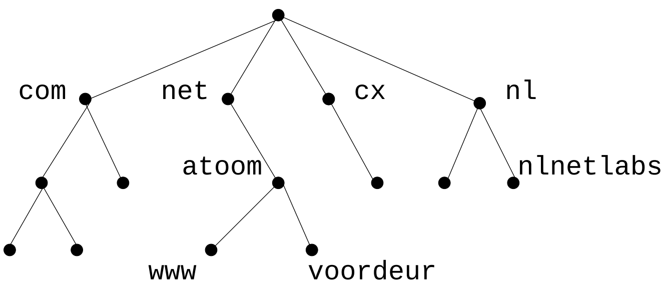

Figure 1: A small part of the DNS-tree.

2.1.1 Some definitions

resolver a set of library routines that creates queries and sends them to a nameserver on the Internet and interprets the results.

resource records each resource record contains information about a zone. A resource record is the smallest piece of information in DNS.

zone is a set of resource records that describe part of the tree.

authoritative nameserver the server that actually runs the software that controls one or more zones.

forwarding/caching nameserver caches queries and does not control a zone.

2.1.2 Caching

The number one priority in the design of DNS was speed. If someone would ask for information concerning a zone in the United States from Europe, a request would have to travel across the ocean get the data and return back to Europe. And if somebody else should want the same information the request is send again. This not only generates this a lot of traffic, it also puts a strain on the nameserver for the zone in the US. This is why DNS caches information. Suppose we have a setup depicted below. When user A asks for information concerning

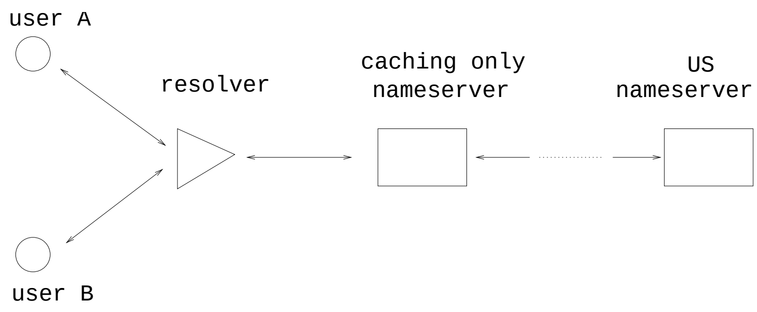

Figure 2: Flow of queries and caching.

the US zone, the request is send via the resolver, through the local nameserver (usually a caching only nameserver) to the US. The information comes back via the same route. The local nameserver will remember this information for a specific time period. A short time later user B asks for exactly the same information. Again it is send via the resolver to the local nameserver. The local nameserver will not forward the request to the US, because it already knows the answer from its cache. Thus bandwidth and time is saved. This is exactly like the memory caching mechanisms found in computers. The users are the requests for data. The resolver is the microcode on the CPU to handle these requests. The local nameserver is the level 1 cache on the CPU, and the authoritative nameserver in the US is comparable with the main memory (RAM).

2.1.3 Zone files

Each (sub)domain name points to information about that (sub)domain. That information is held in a *zone file*. Below is the (current, slightly modified) one for example.com:

```
example.com.   IN  SOA   venera.isi.edu. iana.isi.edu. (
         950301     ; serial
         12H        ; refresh
         1H         ; retry
         2W         ; expire
         1D )       ; minimum

    IN   NS    ns.isi.edu.
    IN   NS    venera.isi.edu.
    IN   MX    10 mail.example.com
    IN   MX    50 mail-backup.example.com

www.example.com.         IN    A     128.9.176.32
mail.example.com.        IN    A     128.9.176.32
mail-backup.example.com. IN    A     128.10.176.32
```

This zone file contains several *resource records* (RR) for example.com. Resource records hold information about a specific domain. The file starts with a *SOA* record, SOA stands for Start Of Authority, this means that the information specified in the records is authoritative for that zone. All the resource records are specified in [6], I will list the most important ones here briefly.

owner which is the domain name where the RR is found.

class for normal DNS operation only the **IN** (Internet) class is used.

type which is an encoded 16 bit value that specifies the type of the resource in this resource record.

- **A** a host address.
- **CNAME** identifies the canonical name of an alias.
- **HINFO** identifies the CPU and OS used by a host.
- **MX** identifies a mail exchange for the domain. See [3] for details.
- **NS** the authoritative name server for the domain.
- **PTR** a pointer to another part of the domain name space.
- **SOA** identifies the start of a zone of authority.

  In the SOA you have the following:

  - **serial** A number to identify the version of the zonefile. Whenever there is a change in the zonefile this number should be increased so that secondary nameservers know that they must get the new information. There is no basic format for this number, but a widely used scheme is: **YYYYMMDDxx**, **xx** is used when there is more than one update on the same day, so the second update on June the second in 2004 is denoted as 2004060202[2].
  - **refresh** this number is the time in seconds in which a secondary nameserver must synchronize its data with the primary nameserver.
  - **retry** if a secondary nameserver can't refresh, it will try it again within this time.
  - **expire** this value is used by secondary nameservers. If the primary nameserver is unavailable, the secondaries will be authoritative for the zone for duration of this time.
  - **minimum** this *was* used as a *time to live* (TTL) value for cached data. When this time expires the cached data is thrown away. Today this value is used as a TTL for negative caching (see [9]). Negative caching is used when a requested domain or a RR does not exist.

In these two lines the nameserver resource records for example.com are defined.

[2] The serial for example.com does not use this format.

```
IN   NS    ns.isi.edu.
IN   NS    venera.isi.edu.
```

The MX record will advertise which host(s) will accept mail for the domain. The numbers 10 and 50 specify the priority for the mail exchange servers. The mailserver with the highest priority (the lowest number) will be tried first, if that server is unavailable the other server(s) are tried in descending priority.

```
IN   MX    10 mail.example.com.
IN   MX    50 mail-backup.example.com.
```

These lines specify some 'A' resource records for different hosts. Further information on the precise format of this file, and DNS in general can be found in [2].

```
www.example.com.          IN    A     128.9.176.32
mail.example.com.         IN    A     128.9.176.32
mail-backup.example.com.  IN    A     128.10.176.32
```

2.2 DNS lookups

A lookup in DNS consist of traversing the tree, until the right node (domain) is found. Let's suppose we want to find the IP address (A resource record) of the server www.example.com. In this example I will assume that no information is cached somewhere. The first thing to find is the nameserver of the .com domain, this information is known by the root servers. There are 13 of these root servers spread all over the world. These servers control the dot zone (".") and therefore serve the top level domains (*TLD*), like .com, .net, .org, etc. The IP addresses of these machines are well known. Next the resolvers asks one of those servers for a authoritative nameserver of example.com. In our example this is venera.isi.edu. That server should be able to tell us what the IP address of www.example.com is. If the server is not reachable the resolver will try ns.isi.edu. Our lookup is now complete; either the nameserver knows the IP address or it does not.

To facilitate further requests in the near future our (local) nameserver will cache the IP number of www.example.com and all other information it acquired with this request (e.g. the nameserver for .com) for a certain amount of time (TTL).

2.2.1 DNS answers in detail

A DNS answer consists of five sections:[3]

header specifies that is a response query and carries some other bits.

question carries the query name and other query parameters.

answer carries RR's which directly answer the query.

[3] More details can be found in [6] section 6.2.

authority carries RR's which describe the authoritative servers.

additional carries RR's which may be helpful in using the RR's in the other sections.

Using the **dig** commando shipped with **BIND8**, the different sections are clearly visible:

```
; <<>> DiG 8.2 <<>> www.example.com
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 1
;; QUERY SECTION:
;;      www.example.com, type = A, class = IN

;; ANSWER SECTION:
www.example.com.        1W IN CNAME     VENERA.ISI.EDU.
VENERA.ISI.EDU.         23h44m55s IN A  128.9.176.32

;; AUTHORITY SECTION:
ISI.EDU.                23h44m55s IN NS  east.ISI.EDU.
ISI.EDU.                23h44m55s IN NS  VENERA.ISI.EDU.
ISI.EDU.                23h44m55s IN NS  ns.ISI.EDU.

;; ADDITIONAL SECTION:
east.ISI.EDU.           4d1h56m5s IN A  38.245.76.2

;; Total query time: 163 msec
;; FROM: floep to SERVER: default -- 213.53.69.1
;; WHEN: Thu Dec 21 16:31:29 2000
;; MSG SIZE  sent: 33  rcvd: 143
```

To cut down on processing load and network bandwidth all answers are put in UDP datagram packets. For historical reasons these UDP packets have a maximum size of 512 bytes. There will always be sent one packet of 512 bytes. If there is room left in the packet an additional section may be added with extra information. A resolver may ask for the MX record of a zone and it will not only receive the name of that server but also it's IP address via the additional section. This trick will prevent an entire lookup. This may not seem much, but a typical rootserver gets more than 6000 queries every second[4], and there are a lot of resolvers on the Internet. So it makes very good sense to cut down on the number of queries.

If an answer is too big to fit in one packet, TCP is used, which adds a lot of extra overhead. In DNSSEC which allows KEY RR of 2048 bits this is an

[4] David R. Conrad said in comp.protocols.dns.bind that "[...] the number of queries per second [is] something like 6000 [...] sustained, last I check[ed] (but that was a while ago)".

issue. Currently there is a draft proposal which allows for larger UDP packets in DNS.[27]

2.2.2 Glue records and delegation points

Glue records are address RR's, but their use is somewhat special. They are needed in the following scenario. A TLD nameserver serves for instance the .com domain. Their zonefile contains a list of domains followed by the nameservers authoritative for that domain. Such an entry is called an delegation point. Now consider this delegation:

```
example.com.   IN NS ns1.example.com
```

This delegation states that to find example.com one must ask ns1.example.com. To find ns1.example.com one must look up the delegation for example.com and there one finds ns1.example.com and thus a loop is created. The resolver will never be able to resolve example.com for this reason. To solve this a glue record is needed. The TLD will not only have a delegation point, but will also carry an extra record:

```
ns1.example.com   IN A 192.168.1.1
```

Now a resolver will be able to resolve the nameserver for example.com. The reason that glue records are special is in that this information actually doesn't belong in the parent's zone. This is an oversight in the DNS specification. The whole problem could also be solved by specifying IP addresses as the data in NS RR's.

2.2.3 Lame delegations

A lame delegation is a delegation that is faulty. This can have two causes.

delegation to an unknown host the nameserver does not exist on the Internet.

delegation to the wrong nameserver the nameserver is not authoritative for the domain.

On both cases the domain will not be visible on the Internet. A lame delegation is by far the most common error in DNS, also see [17], section 2.3 and [22].

2.2.4 Wildcard delegations

Wildcards like " * " can also be used in DNS, just like the wildcard use with a shell under Unix (ls *).

2.2.5 Apex

Apex is the top of a zone. It consists of the SOA record, with the nameserver records and possibly also glue records. The child's zone contains and is authoritative over this information. Some parts, notably the NS records and possibly some glue also reside in the parent's zone. Together these form the apex.

2.3 Security

DNS has worked very well for more than fifteen years, and thus far there has been only one big problem with DNS:[5] it is vulnerable to false data. To use DNS in a reasonable secure fashion, one should direct all the queries directly to an authoritative nameserver, and thus not use the caching mechanism of DNS. While this is a security win it will wreck havoc on the performance of DNS.[6] With IP rerouting a malicious person could fake an authoritative nameserver all together, something DNS can't prevent at all (DNSSEC will prevent this attack).

Most lacking in DNS security is that there is no mechanism in place to check the validity of the information a nameserver or resolver receives. It all boils down to the caching mechanism and the additional answer sections in DNS: a nameserver will "believe" everything that is sent to it and it will also cache this information for the duration of the TTL. This in combination with the additional data section is enough to break DNS in a very trivial way. I anticipate that this attack can be done by one person in couple of hours.

The additional data section holds extra data. The problem with this data is that a resolver, or for that matter another nameserver, has no way of telling if the data is to be trusted or not. It may seem strange that a resolver asks for the MX record of example.com and also get an A record for SecureBank.example.com, but it is perfectly legal.

To start an attack, the attacker must have its own DNS in place, say he owns hack.example.com. The attacker sets the TTL value of hack.example.com to a very low value. This will make sure that no data of hack.example.com is cached and that queries for hack.example.com will be handled by the nameserver that is under the attacker's control. Then he chooses an victim, SecureBank.example.com. Next the attacker will forge an A record for SecureBank.example.com and hack his nameserver to return this information in the additional section. The information in the additional section will have a large TTL so that it *will* be cached.

Now the attacker queries an important nameserver on the Internet for information on the hack.example.com zone. Since the TTL for the attackers zone is very low, it is likely that this information is not cached. This means that the request will be forwarded to the attacker's nameserver. This in turn means that the answer the nameserver returns will have the bogus additional section,

[5] There are more problems, like the error prone configuration files and the lack of dynamic updates.

[6] These performance and scalability issues were the exact reasons for DNS in the first place.

and since that TTL is large it will be cached for some time. The attacker will query the important nameserver every few seconds until the TTL of SecureBank.example.com expires. At that moment his bogus data will be considered valid and the IP address of SecureBank is changed.

To create some financial reward for all the work the attacker has put in, he also recreates a part of the site of the bank, so unsuspecting visitors will happily supply there deepest banking secrets to the fake site. With this information he can go to the real site of the bank and rob peoples accounts. The attacker would probably get away with this attack, it is even possible that nobody would even know that such an attack is taking place, because nothing has stopped working. The only thing SecureBank would notice is a slight decrease in traffic. If system administrators search the log files to see if they can find something strange they will find nothing, because there are no errors.

This major flaw in DNS is documented in [1]. This report has been kept secret for almost five years, because of the huge impact it would have. This attack is still possible on the Internet of today. Electronic banking and e-commerce are very dangerous things to do on an Internet without DNSSEC.

3 Secure Domain Name System

To combat the security problems of DNS, DNSSEC was developed. DNSSEC adds cryptographic signatures to the information in DNS, and thus allows the information to be authenticated.

The DNS security extensions define the security services that are incorporated into DNS, these are:

- data origin authentication.
- transaction and request authentication.
- key distribution.

DNSSEC does not provide protection against denial of service attacks. It does not either describe any means for protecting against attacks on the name server itself. It should also be noted that DNSSEC does not assure the correctness of the DNS data in any way. DNSSEC provides for integrity of the data, but the DNS data can still be misconfigured at the name server.[11]

DNSSEC introduces the concepts of secure and insecure zones. A secure zone will have at least one signature resource record for every resource under that zone. The only exceptions to this rule are glue records and delegation point NS resource records. These records are signed in the child zone and the signature are in the child zonefile a not in the parent zonefile.[11, 16]

3.1 DNSSEC design and structure

The structure of DNSSEC is the same as the one from DNS. DNSSEC adds a few enhancements to DNS.

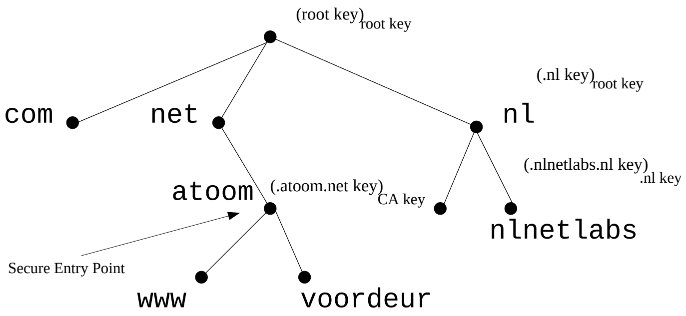

Figure 3: A small part of the DNSSEC-tree. The $(x)_y$ notation denotes that key x is signed by key y.

The DNS Security extensions introduces a number of new records. These include

signature (SIG) resource record defined to store signatures in the DNS. It provides a cryptographical binding of records in a resource record set being signed to the signer and a validity interval. The signature is created using public key cryptography. If a server supports DNSSEC and is thus security aware, it will attempt to return the relevant RR's and the corresponding SIG records in an answer to a query.

The SIG resource record contains information about the algorithm used, validity interval, the signer name and the actual signature. The algorithm field specifies the algorithm to be used. Currently RSA and DSA can be used for creating signatures. A special value is reserved for being able to use private algorithms locally. A value is also reserved for use with elliptic curve cryptography. The signer name is the domain name of the signer that generates the signature. The actual signature ensures the integrity of both the RRset signed and the fields in the signature record.[16, 11]

KEY resource record used to store the public part of a private/public keypair. The key is associated with the DNS name. The owner of the key can be a DNS zone, a user, a host or some other end entity. A resource record with the same name and same type can be associated with several key records. The key RR is authenticated by a SIG RR like other DNS resource records. The key resource record contains the key, the algorithm the key is to be used with, and a value indicating the protocols the key is to be used with. In addition to the algorithms used by the SIG RR, Diffie-Hellman keys can be stored in a key RR. The protocol field currently makes it possible to bind the key for use with TLS, e-mail, DNSSEC and IPSEC. A range of values are reserved for new protocols to be added in the future. The protocol field can also indicate that the key can be used with all of the protocols.[16, 11]

NXT resource record the purpose is to be able to provide data origin authentication of a non-existent name or the non-existence of a certain resource record type for a DNS name. The owner of a NXT record is an existing DNS name in the zone. The NXT resource record contains the name of the next name in the zone, thus stating that there can be no resource records between the owner name and the next name. To be able to use the concept of a next name in the DNS implies that the records in a zone have to be canonically ordered. The last NXT record in a zone will contain the zone name, treating the name space as circular. At a delegation point, there will be two NXT records for the same name, one residing above the zone cut and one residing below the zone cut. As with all records, the NXT records are authenticated by a SIG record.[16, 11]

CERT resource record used to store certificates in the DNS. The CERT resource record contains information about the algorithm used, the type of the certificate and the actual certificate. In addition to the algorithms used in KEY and SIG records, the algorithm field can indicate that the certificate is to be used with an algorithm unknown to DNSSEC. This makes it possible to use the certificate with algorithms that are not standardized for DNSSEC. The types of certificates currently defined are X.509, SPKI and PGP certificates. One type is reserved to indicate a certificate format defined by an absolute URI. This URI will contain documentation about the format of the certificate. Type values are reserved for future IANA assignment of new types of certificates. The certificate section contains the actual certificate data represented in base 64. The section can also include a certificate revocation list.

As the CERT record can contain a certificate, it is possible to to use DNS for storage of public keys. It is intended that personal public keys should be stored in the DNS using the CERT record, and not by using the KEY record.[16, 18]

As with all the other RR in the DNS these records are also cached. To start using DNSSEC one must create a public and private key pair. The private key is used to sign ones zonefile e.g.. example.com. The public key is signed by the parent (in this case .com) and that signed key is then inserted in the apex of example.com. In this way the parent will tell the world that it trusts it's child.

If DNS is completely replaced by DNSSEC, there will be a few keys that every resolver knows, comparable with the knowledge of the IP addresses of the root servers. the rootservers and will use that as a starting point. The resolver in DNSSEC is preconfigured with one or more root keys. Those keys cannot be delivered with DNSSEC, they must come with the resolver and the administrator should check if the keys are genuine. A SIG RR contains a field (see paragraph 3.3) with information on whose key was used to create the SIG. That key is also signed by someone else, and so on and so on. A resolver must follow this chain up to a known key that is preconfigured. When it reaches this point it will declare all the signatures it found valid and the information is authenticated. In other words: "The resolver walks the chain of trust".

3.2 Key RR's in detail

This is how the key RR is defined:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|             flags             |    protocol   |   algorithm   |
+---------------------------------------------------------------+
|                                                               /
/                          public key                           /
/                                                               /
+---------------------------------------------------------------|
```

Figure 4: Definition of bits in the key resource record.

The different fields in the RR have the following meaning.

flags sets a few options for the key

protocol in the future these keys could be used for other things besides DNSSEC, like secure email or IPSEC

algorithm which algorithm is used to create the key, RSA/MD5, DiffieHellman and DSA or an algorithm defined in [11]

3.3 SIG RR's in detail

The SIG RR is more complicated than the key RR. RFC 2535 [11] states

> The SIG or "signature" resource record (RR) is the fundamental way that data is authenticated in the secure Domain Name System (DNS). As such it is the heart of the security provided.
>
> The SIG RR unforgeable authenticates an RRset of a particular type, class, and name and binds it to a time interval and the signer's domain name. This is done using cryptographic techniques and the signer's private key. The signer is frequently the owner of the zone from which the RR originated.

The RR itself is defined in figure 5. The different fields are defined as:

type covered to which RR belongs this SIG to

algorithm the same as the algorithm field in the key RR

labels how many labels has the owner name; these can be calculated from a domain name by counting the number of dots + 1, so example.com has two labels

```
                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|        type covered           | algorithm     |      labels   |
+---------------------------------------------------------------+
|                         original TTL                          |
+---------------------------------------------------------------+
|                      signature expiration                     |
+---------------------------------------------------------------+
|                      signature inception                      |
+---------------------------------------------------------------+
|            key  tag           |                               |
+-------------------------------+        signer's name          +
|                                                               /
+--------------------------------------------------------------/
/                                                              /
/                            signature                         /
/                                                              /
+---------------------------------------------------------------+
```

Figure 5: Definition of bits in the signature resource record.

original TTL when verifying a signature the original TTL must be known, which was used at the signature's creation time

signature expiration a signature has a lifetime extending from the inception time to the expiration time. When the expiration time is reached the signature is no longer valid.

signature inception start date of the validity of a signature

key tag there may be multiple keys that can be used to check if the signature is valid, with this **key tag** one can select which one is to be used.

signer's name domain name of signer generating the SIG RR.

signature the actual signature

3.4 How DNSSEC works

By using DNSSEC, one is building a chain of trust. The chain is created by letting a parent sign a child's public key. The chains starts with a key that is known to a resolver. Ideally this key would be the key of the rootservers of the Internet. This key could be published in a national newspaper and published on a website, so the everyone who wishes can check the correctness of the parent's key.

If a resolver trusts a TLD, by having that key preconfigured, that point in the DNSSEC tree is called a *secure entry point* (see figure 3 on page 15). In the ideal case there is only one secure entry point, the rootservers (".").

If a resolver finds a signature made with a key it doesn't know, it will go up the chain to look for a key it does know. The resolvers eventually finds a key it trust or it doesn't. In the first case the results can be validated, and either be found secure or insecure, in the latter case the results are considered bad. To prevent loops from occurring, BIND9 only allows subsequent keys from zones above the current zone, thus a search for a non existing key will end at ".". This was also something that was not clear from the DNSSEC specification.

It is this extension of trust that makes DNSSEC work. The main problem that remains is establishing an initial form of trust. If DNSSEC is widely employed on the Internet it promises a universal key distribution mechanism, something a lot of people eagerly await.

3.5 DNSSEC lookups

A resolver will have to know beforehand if the information it is looking up, will be secure or not. If not, such a zone is vulnerable to zonejacking. How does a resolver determine it is entering a secure zone? It will use the preconfigured key as a starting point. If a resolver has a key for the .com zone, it will know that every domain under .com is to be considered verifiable secure, verifiable insecure or bad.

Let's look again at www.example.com, but now in the case of DNSSEC. We want to know the IP address of that host. First determine the security status of example.com. If we have a key for .com or example.com, we know we are entering a secure zone, if we don't have that key it is safe to assume that example.com is not secured and not running DNSSEC, the lookup will then be similar to the lookup in paragraph 2.2. Assume the resolver has the key for .com. It will start the search from that point, a secure entry point. For example.com there must be a key present in the .com zone indicating the security status of example.com. When this key is found, it's signature can be checked with the key from .com. If this check fails the resolver either has the wrong key for .com or something more sinister is going on. In both cases the resolver can not trust anything originating from .com and the query stops. If this key record for example.com is not found at .com something is very wrong (probably human intervention is needed at .com). Next the resolvers checks the contents of the example.com key. If the key is empty (a so called *null key*) example.com is considered verifiable insecure. The lookup will then proceed as a normal DNS lookup. If the key is not empty example.com is considered secure. This does *not* mean that example.com is verifiable secure, it still can be bad. Next the resolver will "enter" the delegation; it will go to the nameservers of example.com and it will ask an A RR for www.example.com. If example.com is verifiable insecure there will be no signature to check, so the resolver is finished in this case. In the secure case, the resolver will not only receive the A record but also a SIG record that it now can check with the key from example.com. If the signature is OK the resolver is finished, if not, example.com is marked bad and all records are dropped.

3.5.1 Negative answers

With DNS a query for a non existent host or zone will result in a NXDOMAIN (Non Existing) answer. With DNSSEC this is a bit more complicated. For practical reasons a zone in DNSSEC is not signed on the fly. It is signed and then loaded on the nameserver. So how do you handle requests for information you don't have? One could make an exception for negative answers and allow them to be signed on the fly, but that is not the approach that is chosen in DNSSEC, this would make a denial of service attack on a nameserver to trivial. With DNSSEC a zone is ordered when it is signed and every DNS name in a zone gets a NXT record which points to the next name in the zone. The last NXT record points back to the first. These NXT can be signed. Instead of sending a NXDOMAIN DNSSEC sends back a NXT record.

3.5.2 Stopping attacks

DNSSEC stops zone hijacking (sometimes called zone jacking). Attacks as described in paragraph 2.3 are impossible. Suppose the attacker wants to hijack a secure zone. A secure zone is signed with the private key of that zone and the accompanying public key is signed by its parent. An attacker will have to set up a zone on it's own nameserver and will have to sign it. Since it does not know the private key of the zone to be hijacked, he cannot do that. If he makes his own private key and tries to poison other nameservers they will not accept the data he is sending because the signatures will not check out, and the data is dropped (and also not cached).

3.6 Consequences of DNSSEC

3.6.1 Increased zone sizes

Signing a zone file causes it to increase in size. There is the addition of a key RR and the doubling of the number of RR's, because every RR now must have a signature RR and a NXT RR. They typically increase by a factor of 2-5. Obviously this has scalability implications for large zones (especially .com, .org and .net). Although this is a big increase it is a problem that is present even without DNSSEC. The .com is growing a at rate of 1 million each month, so TLD or gTLD and even ccTLD will be forced to think about solution to get a grip on such big zone files. Although few public publications exist on this subject there are a number of solutions, directly using a database with a nameserver is one of them. In fact, BIND9 has hooks in its code to use a database as back end.

3.6.2 More administrator work

DNSSEC would increase the amount of attention paid to the DNS. Authenticated data would "expire" from the DNS and would need to be re-validated on a regular basis. Not only would this require more resources (CPU, disk and bandwidth), but the registries and registrars would have a recurring relationship rather than the present "register and forget". This will require new operating procedures.

3.6.3 Zone walking

NXT records make it possible to do a zone transfer by walking the chain of NXT records. This works as follows. Ask a name to the nameserver of which you are certain it does not exist. You will then be given a NXT record back pointing to the next name that does exist for that zone. Now ask the NXT record of that name, then ask the NXT of that answer and so on, until you encounter a name you already have. You now know the complete zone and you have effectively done a complete zone transfer. Whether this is really an issue (security by obscurity) is still under debate, but there is an effort underway to use NO records [25], which returns a hash value of the next record instead of the next record itself. With NO records it is also possible to use one NO record for multiple other RR, this can be used to reduce the size of the zone files in DNSSEC.

4 How will DNSSEC be organized?

For the security to work in DNSSEC, a strict organization is needed between the various parties involved. The work for NLnet Labs and therefore also in the master thesis has concentrated on the implementation at the TLD level.

To ease the implementation it is best when this organization matches the current organization closely. In DNS there are usually four parties involved when claiming a domain under a TLD, and now also when making that domain secure.

domain-holder the owner of dom.top. Every domain has a domain-holder.

zone-administrator hostmaster of dom.top. Takes care of the technical side. Multiple domains can be handled by one administrator. The zone administrator has no formal role.

registrar service provider for dom.top. Most TLD have multiple registrars. This helps to avoid unwanted monopolies and makes the organization more scalable.

registry owner and hostmaster for top. There is only one registry for top.

All of these parties must communicate with each other. For DNSSEC it is crucial that the parent signs the *right* key and can not be tricked to sign a key for a fake zone. It must be noted that the registry only wants to talk to the registrars. This helps to reduce the load on the registry.

Although the zone administrator plays an important role is DNS and thus in DNSSEC it is important to see that this role is purely technical. A zone administrator does technical work and assumes no formal responsibility for that work. In the two figures below the zone administrator sits under another party, this means that the party above it is formally responsible for the zone administrator.

In the Netherlands the parties involved can be mapped as follows. The domain-holder is someone who owns a business.[7] The registrar is an Internet provider(ISP, the zone-administrator works for the same provider and the registry is SIDN. This is the configuration shown below. The second figure is a different configuration, in which the holder does its own DNS administration, although it may have been outsourced.

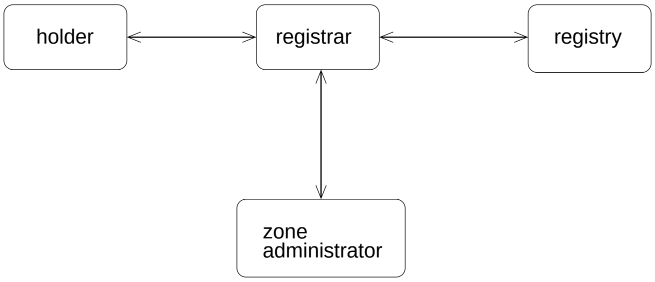

Figure 6: Communication between the parties. The registrar does some technical work for the domainholder.

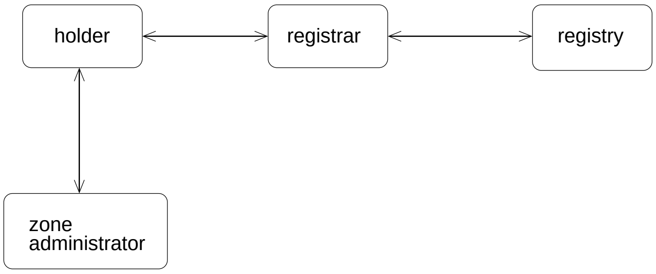

Figure 7: Communication between the parties. The holder takes care of the zone administration.

[7]On the 15^{th} of November 2000 privately owned domains were allowed.

4.1 Overlapping parties

There are some situations in which parties overlap or are the same entity. In both configurations shown above this overlap is possible. In this four party model there are 4 possibilities:

1. holder and zone administrator overlap.
   Think of big corporations or scientific organizations (universities) who have enough knowledge to do their own DNS and zone administration.

2. zone administrator and registrar overlap.
   Usual for "hosting providers".

3. holder and registrar overlap.
   This could be an organization that can do the registration but they have outsourced the actual DNS maintenance, for instance an advertising agency.

4. registrar and registry overlap.
   The .com was organized like that and there are still some ccTLD that use this scheme.

The more parties overlap, the easier the communication will get.

4.2 Establishing trust

In section 5 I will describe a method for exchanging a public encryption key with a domain registry, so that the registry can sign the public key with its one private key and can send the resulting signature back the owner of the public key. The crucial problem is to make sure that the public key the registry is signing is really from the person that is authoritative for the domain. The other steps in the model are less crucial; when the key is signed that signature is meant to be public data anyway. The method must be as secure as possible and also be *very* scalable.

4.2.1 Four parties

In this case all four parties have to communicate with each other. In some cases the process of registration is informal and there is little or no communication. Implementing DNSSEC will then be difficult, however in many cases there already exists some communication infrastructure between the four: the domain is registered and active. This is very important to note: There has been some form of communication between the parties.

Suppose that the registry already has DNSSEC in place and they are secure. Much – if not all – of the communication will take place via email. In this example I will assume that there isn't any overlap between the different parties and that the configuration shown in figure 7 is in use. Figure 9 depicts the step and the flow of the keys when becoming secure in DNSSEC. Figure 8 depicts the flow of keys when configuration in figure 6 is used.

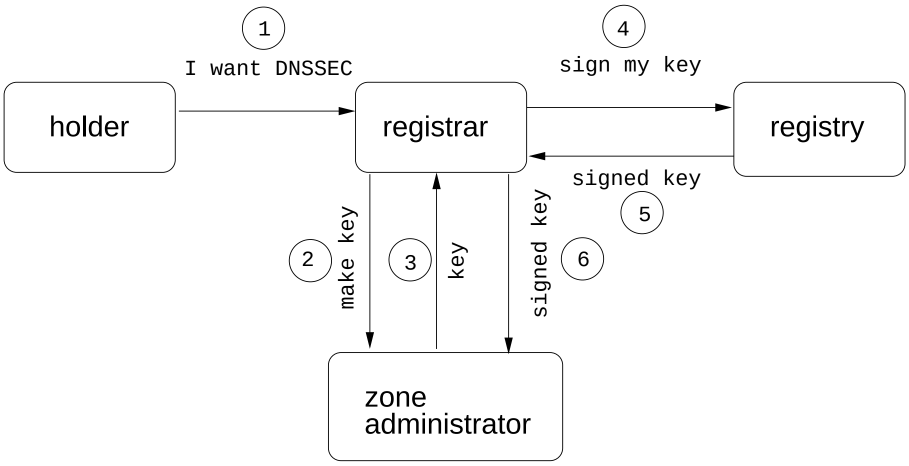

Figure 8: Communication when becoming secure, administrator works for registrar.

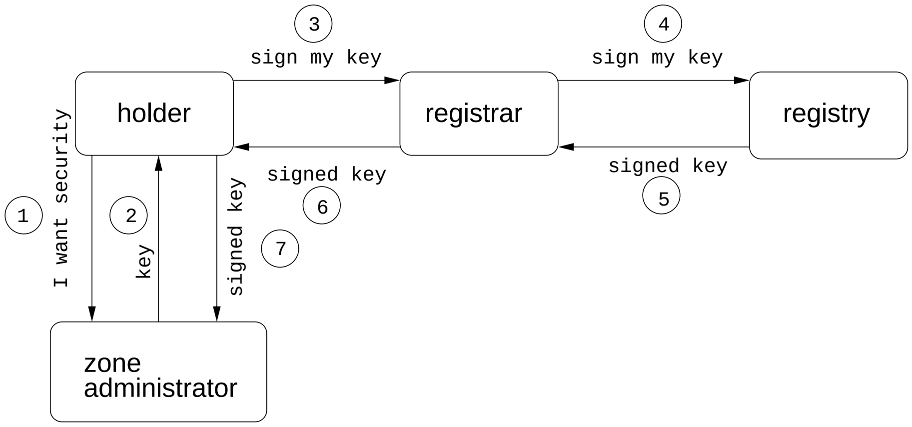

Figure 9: Communication when becoming secure, administrator works for holder.

4.3 Becoming secure

It starts with the desire of the domain-holder to become secure. The zone administrator must be told about this desire. This is the first step. In almost all cases there will already be a means of communication between the two (direct or via the registrar).

The next step in the process for the administrator is to produce a private- and public-key via a public/private key encryption algorithm.

Then the public key must be sent to the registry. Because the registry only wants to deal with its registrars, this request has to go through the registrar. Depending on the configuration in use (see figure 9 or 8) this communication either goes through the holder or is direct.

4.3.1 Signing at the registry

The key has arrived at the registry. The registry trusts the registrar, so it can assume that the key signing request is legitimate. Note that it is up to the registrar to make sure the requesting party really is the holder of the domain. If it turns out not to be legitimate, the registry must be able to hold the registrar legally responsible.

The registry now signs the key to create a signature that can be sent back to the registrar.

To further reduce the possibility of being tricked by a fake domainholder, the registry informs the admin-c and tech-c for the domain that is to be secured.

As soon as the null key in the zone's apex is replaced with the new key the child's zone is considered secure. If the child's zone is not signed with that key, the zone will be marked as bad and will drop of the Internet.

4.3.2 Administrator works for registrar

If the administrator works for the registrar and not for the domain holder, figure 6 on page 22 not much is changing, except that the domainholder waves its rights for its own security. The security of the domain is entirely in the hands of the registrar. In this case the domain holder will not even have direct access to its private key.

4.3.3 Overlapping parties

When parties overlap the communication scheme will get more simple. It is just a matter of scaling down the four parties scheme.

4.4 Maintaining security

Now that the first key is exchanged, it can be used to sign future requests. Section 6 describes the procedures for maintaining a zonefile in a registry where all the parties are authenticated and section 5 delves deeper in the subject of authentication.

5 DNSSEC authentication mechanism

A key exchange must start somewhere. This is also the case with DNSSEC. This first step to establish trust is the *most crucial step* in DNSSEC. If an attacker can somehow get involved in this first step the whole zone can be controlled by the attacker and the whole purpose of running DNSSEC is defeated.

A scheme must be easy to implement, fast, scalable and reasonable secure against attacks. Because of the sheer number of zones a typical TLD has to deal with the level of out of band communication between the parties must also be kept as low as possible. In the ideal case all communication will take place using existing DNS(SEC) messaging techniques.

If initial trust is established and the parties involved are confident that everything went all right, those keys can be used for the further communication between the two.

5.1 Authentication mechanism I

The scheme presented in this paragraph should only be used as a last resort. This scheme is only needed when performing the crucial step: the delivery of the *right* key from the *right* person to the registrar/registry.

When a party receives a request to sign a key, this request is delivered via email. To get as much information one should do the following:

- archive the mail.
- do a traceroute to the domain and also archive that information.

It then sends an email back to the responsible person for the domain with the question if the request was genuine.

Three things can happen at this point:

1. the registrar/registry receives an answer that nobody requested to be secure. The request is then simple dropped.
2. there is a positive answer.
3. there is no answer or the mail bounces. The request is then also dropped.

If the receiving party is forced to drop a key signing request due to failures, it essentially does nothing with the possibly faked request. It is entirely up to requesting party to decide if they want to investigate this incident further.

5.1.1 Levels of trust

Scheme I should be scalable because much of it can be automated, but it is not as secure as it should be. To hand over the public key of a domain other methods are also possible. I call the different methods levels of trust as each level is more secure than the former, but also is more difficult to automate.

Level 0 this is scheme 1. This could be useful for a large part of the .com domain.

Level 1 the registrar or registry does not want to use a plain text email based authentication. The email will be signed with PGP. It still will be necessary to establish a starting point for trust. The registrar or registry can put their public key on a website or store it in one of the PKI servers around the world. Assuming that a registrar wants to setup a secure communication with a domain holder. The domain holder can get to the key of the registrar. Now the registrar wants to have the public key of the holder. If the holder also has a PGP key in a PKI server the registrar can check if it trusts someone that signed the PGP key of the holder. Otherwise the holder has to submit its PGP key to the registry and the registry must then take another step to verify the key. What they can do is for instance call the domain holder by telephone and verify the PGP key in this way.

When those PGP keys are exchanged they have authenticated each other and can proceed with the securing of the holder's domain.

Two things must be noted with this level. For one, some PKI's are not as secure as we think, see [26]. And second, calling someone to verify its key is very expensive. Consider a large TLD, with 1 million soon to be secured delegations. If they all want to use level 1, this could mean 1 million phone calls. Each call will take up to 5 minutes (checking keys is tedious), so this will amount to 5 million minutes (some 83 thousand hours or 500 man months) of phone calls. This problem could maybe be solved by using a call center.

Level 2 essentially the same scheme as in level 1. The verifying step now consists of the domain holder actually coming by at the holder to identify himself and to bind his PGP key to holder's identity. If a TLD has a lot of holders that want to be secured and want to use level 2, this scheme is going to be impracticable. Imagine 1 million people coming by the registrar to prove their identity.

5.2 Authentication mechanism II

Almost always a domain is already registered and then it will be made secure. The infrastructure to register is in place and can be used. The idea here is to use that infrastructure to go from unregistered and insecure to registered and secure. This will also solve problems described above.

The child is a registered domain under a TLD or ccTLD. If the child does not run a secure aware DNS server, it must upgrade to one. The following steps have to be taken to become secure.

1. the child generates a public key pair.
2. the child places the public key in its zone file.
3. the child signals its parent that it is ready to become secure.
4. the parent

- looks up the IP numbers of the authoritative nameservers of the child. This done by checking the TLD (zone)database[8], *not* by using the DNS, which can be spoofed.
- then does a zone transfer of the child's zone, directly using the authoritative nameservers.
- does a traceroute to the child's zone.

5. Depending on whether there is a key in the transferred zone, the parent either drops the request or signs the key and informs the tech-c and admin-c (e.g. registrant and domainholder) of this step.

6. the parent puts the signature over the child's key in its zonefile.

7. the parent signals the child that it is now secure.

Figure 10 gives a graphical representation of these steps. The parent is

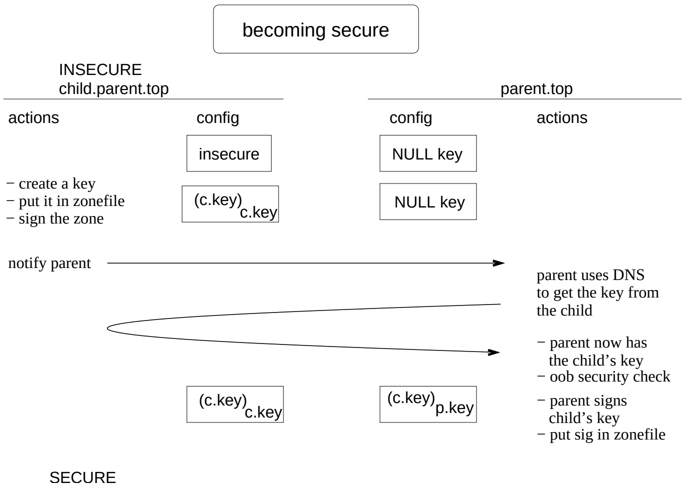

Figure 10: Initial key exchange.

now in the possession of the public key belonging to child's zone. The parent's public key should be widely known. With these public keys all all further communication can be signed.

The nice thing about this scheme is that is utilizes the DNS in the most secure manner to allow a smooth upgrade to DNSSEC. It is however still susceptible for IP rerouting attacks, with DNS it is impossible to defend against such attacks, but with the traceroute information gathered step 4, it may be possible to find the attacker.

[8]This same database is used to create the zonefile.

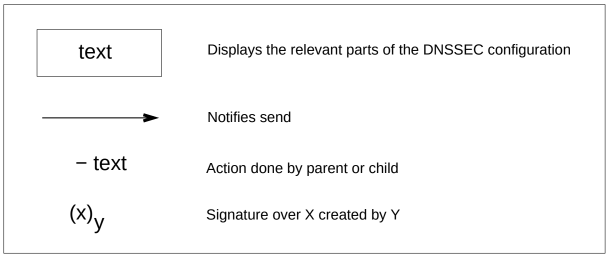

Figure 11: Legend for figures 10, 12, 13 and 14

BIND9 currently doesn't support this model. With BIND9 a key is signed with itself and the resulting keyset (the key and the signature of that key) is send to the registry. This extra step is taken to uphold integrity of the key that is send to the registry.

6 Signing of zones

With DNSSEC zones need to be signed in order to create the SIG resource records. Every zone in DNSSEC is signed with the private key belonging to that zone. The KEY RR of a zone is not signed with the private key belonging to that but with a private key belonging to somebody else. This could be the parent zone or a completely different domain somewhere on the Internet, like for instance verisign.com. This record is then stored in that "higher" zone.

Every zone signs its own zonefile. If .com signs its zone, it doesn't drops down to also sign the zones it is delegating, .com cannot even access the child zonefiles.

6.1 On-tree signing

When the parent of a zone signs the key for that child zone, this is called on-tree signing. This is the primary way to set up a secure DNS tree.

6.2 Off-tree signing

This is another possibility. A zone could be signed by another zone, which is *not* the parent. Although the RFC [11] does not specify any restrictions on the signing zone, BIND9 only allows zones that are on the same level or higher in the DNS tree, this will make sure that there are no loops created.

It is important to note that with off-tree signing the tree structure of DNS is bypassed, hence the name off-tree. This not without problems. Consider the example given in paragraph 3.5 again. As stated a resolver will have to know beforehand if a zone is secure or not. It does this by having a key preconfigured, which it trusts. With on-tree signing it simply follows the DNS structure downwards until the right zone is reached, which then is validated or invalidated. With off-tree signing the resolver cannot simple follow down the DNS tree, because that structure is bypassed when using off-tree signing. Thus it has no way of telling if the domain it is resolving is using DNSSEC or not. Thus DNSSEC seems to be impossible with zones that are signed off-tree. Some companies think there is big business in off-tree signing.

Off-tree signing seems only possible if the zone that is using it is also is a well known secure entry point. This would mean that if off-tree signing ever gets popular quite a number of keys should be preconfigured at the resolvers. But it is advisable to limit the number of keys that should be preconfigured at the resolver level. If those keys aren't in sync with the actual keys used on the Internet entire TLD's may be marked bad by the resolver and are discarded.

The .nl.nl test bed should not allow for third party signed zones under its delegation.

6.3 Location of key signatures

RFC 2535 [11] states that a zone key must be present in the apex of a zone. This can be at the delegation point in the parent's zonefile (normally the case for null keys), or in the child's zonefile, or in both. This key must also be signed by the parent, so there is also the question where this signature is located.

The original idea was to have the KEY RR and the parent's SIG to reside in the child's zone and perhaps also in the parent's zone. There is a draft proposal [24], that describes how a keyrollover can be handled.

At NLnet Labs we found that storing the parent's signature over the child's key in the child's zone:

- makes resigning a child's KEY difficult.
- makes a scheduled parent-keyrollover very complicated.
- makes an unscheduled keyrollover virtually impossible.

6.3.1 NLnet Labs proposal

NLnet Labs has proposed an alternative scheme in which the parent's sig over the child's key is *only* stored in the parent's zone. This would solve the above problems. Currently this idea is a proposal for a draft RFC.

The exact proposal is:

- the child's zonefile contains its zonekey, signed by itself (selfsigned).
- in the parent's zonefile the child's zonekey is also located, now signed with the parent's zone key.

The rest of this section deals with this proposal and describes the resulting procedures for a TLD.

6.4 Resigning a TLD

Resigning a zone is necessary before the current signatures expire. When all the SIG records, produced by the TLD's zone key are kept in the TLD's zonefile, and only there, such a resign session is trivial, as only one party (the TLD) and one zonefile is involved.

Note, when SIG records are (also) included in child zonefiles, special procedures are needed to take care of properly and timely updating also these zonefiles to prevent these children from being (temporarily) bad. As one can see in figure 12 resigning a TLD using our scheme is rather trivial.

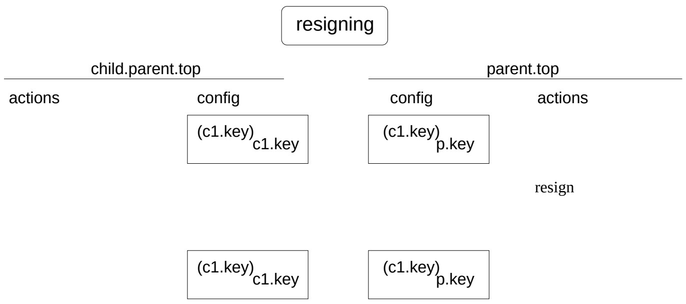

Figure 12: Resigning a TLD.

6.5 Key rollovers

Private keys can be stolen or a key can become over used.[9] In both cases a new a new key must be signed and distributed. This event is called keyrollover. We further distinguish between a *scheduled* and an *unscheduled* key rollover. A scheduled rollover is announced before hand. An unscheduled key rollover is needed when a private key is compromised.

6.5.1 Scheduled keyrollover

When the signatures, produced by the key to be rolled over, are all in one zone file (NLnet Labs proposal), there are two parties involved.

Let us look at an example where a TLD rolls over its zone key. The new key needs to be signed with the root's key before it can be used to sign the TLD zone and the zone keys of the TLD's children. The steps that need to be taken by TLD and root are: [10]

[9] When a lot of data is encrypted/signed with a key it becomes more feasible to calculate the private key from the public using the encrypted data, thus it makes sense to use new keys after some time.

[10] This is a four-step procedure, which can easily be automated. This same procedure can be used anywhere else in the DNS-tree.

1. the TLD adds the new key to its keyset in its zonefile. This zone and keyset are signed with the old zonekey. Then the TLD signals the parent

2. the root copies the new keyset, consisting of the new and the old key, in its zonefile, resigns it and signals the TLD

3. the TLD removes the old key from its keyset, resigns its zone with the new key, and signals the the root

4. The root copies the new keyset, now consisting of the new key only, and resigns it

Figure 13 gives a graphical representation of these steps.

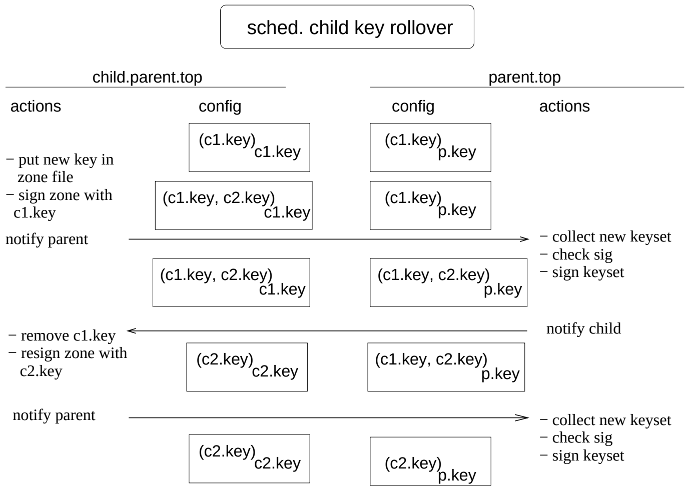

Figure 13: Scheduled key rollover.

Note, when these SIGs, are (also) included in the children zonefiles, there are three generations and thus many more parties and zonefiles involved. For a large TLD this means many millions of parties and zonefiles. As now three generations are involved, the synchronization requires at least seven steps. One step involves waiting for all child-zones to update their zonefiles.

6.5.2 Unscheduled keyrollover

Although nobody hopes that this will ever happen, we must be able to cope with possible key compromises. When such an event occurs, an immediate keyrollover is needed and must be completed in the shortest possible time.

With two parties involved, it will still be awkward, but not impossible to update two zonefiles overnight. “Out-of-band” communication between the two parties will be necessary also, since the compromised old key can not be trusted. We think that between two parties this is doable.

With the update of millions of zonefiles, and overnight out-of-band communication and synchronization between three generations, doom-scenarios as "reboot of the Internet"[11] come to mind. The procedure from Andrews and Eastlake is simply impossible in such an event. Figure 14 depicts the steps needed in this event.

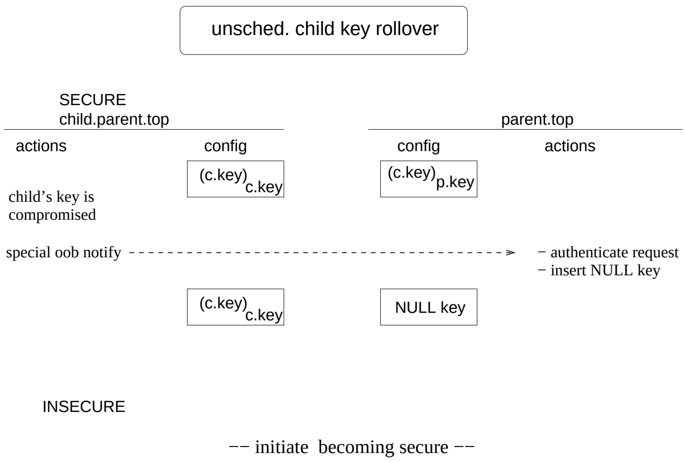

Figure 14: Unscheduled key rollover.

6.5.3 Possible problems

A child could be victimized unknowingly by a parent who's key is compromised or by an error. We have distinguished the following scenarios and adapted our procedure scenarios where necessary:

1. an insecure child. In this case, the child is always at the mercy of its parent.

2. the request is screwed up and the parent signs and includes a malformed key in its zone. With the key-rollover procedure we propose (and describe above) the child could and should check for this before taking the new key into production.

3. a malicious person asks the parent to sign a false key with the intend to hijack the zone. When the child signs the new key with its old key as we propose, this should not be possible. Of course, the parent should check the old key's signature as part of the procedure.

[11] What happens then is that all the nameservers on the Internet, will be stopped for some time and then will be restarted again and that is something that is comparable with a reboot of a PC.

Storing the parent's SIG over the child's key in the parent's zonefile simplifies the communication issues enormously. For a simple resigning, no communication is needed, for a key-rollover only two parties are involved. When this SIG is (also) included in the child's zonefile, simple resigning involves all children and a key-rollover involves three generations.

We proposed this idea at the 49^{th} IETF meeting in San Diego. The DNS-EXT working group liked the idea but they pointed out that it violated one of the base principles of DNS. All information about a zone must be found at that zone, and that is not the case if the SIGs are stored at the parent. We feel that DNSSEC is important enough to allow for this change, but it remains to be seen if the IETF agrees. Keep in mind that the null key *is* stored in the parent's zone, although it defines the (in)security of a child zones. This is an inconsistency in RFC 2535[11].

It must also be noted that a key rollover means that at one point in the time the key must be replaced with a new key. At that moment those keys are cached in nameservers around the world. If a nameserver has cached a key of a zone and not the corresponding data there is a problem. If it retrieves the data from the authoritative server that data is signed with the new key, but that nameserver thinks it already has that key. This will mean that the data does not have a valid signature and is discarded, marking the zone as bad. Until the TTL of the key is expired this situation will exist. There is currently no keyrollover scheme that can deal with this problem. It is expected that this must be solved at the resolver level.

7 Scalability

7.1 Signing of large zones

At the beginning of 2000 there was very limited data on the duration of a zone file signing session of large zones. Large TLD's were very concerned that signing of their zone would take an impractical amount of time (weeks or more).

We had access to the German zone file that consisted of about two million domains. NLnet Labs had already demonstrated that it is was feasible to sign the German (.de) zone file [13]. The largest zone in the DNS is the .com zone which is administrated by the InterNIC. At first we could not get the actual .com zone, because InterNIC was rather reluctant to just give us the .com zone, so we decided to construct a very large zone ourselves, using the .de zone.

The hardware we used for these experiments consisted of a $ 5000 Alpha DS10 from Compaq. The machine had a total of 20 GB of Memory (256 MB RAM, the rest swap). The operating system was Linux (RedHat6.2 for Alpha).

7.1.1 German TLD zone

For the signing we used BIND9.0.0b5, the latest version of BIND9 available at the time. We converted the .de zone to a delegation only zone. We also stripped the comments in the file. After that the file consisted of a small header and for every domain two nameservers, e.g.:

```
a--a NS enterprise.capcom
a--a NS ns.a--a
```

Because the .de zone was too small, we decided to make it 8 times as big. With an (**awk**) script we created 7 files consisting of different domains all in .de zone.

7.1.2 The numbers

The original file consisted of 4,060,720 NS records, the generated file was a little bit shorter, they had 4,060,648 NS records. The new .de file was build as follows:

```
$TTL 1d
$ORIGIN de.
@ IN SOA dns.denic hostmaster.denic 2000032901 3h 2h 41d16h 1d

$INCLUDE Kde.+003+38805.key

NS admii.arl.mil.
NS auth03.ns.de.uu.net.
NS auth61.ns.uu.net.
NS dns.denic
NS ns.ripe.net.
NS sunic.sunet.se.
NS dns.nic.xlink.net.

$INCLUDE 00de.
$INCLUDE a1de.
$INCLUDE b1de.
$INCLUDE c1de.
$INCLUDE d1de.
$INCLUDE e1de.
$INCLUDE f1de.
$INCLUDE g1de.
```

Every domain has two NS records, this would give us a total of $16,242,628$ domains, more than the .com zone on the 9^{th} august 2000 had.[12] Next we started the signing process with a 768 bit DSA key. After about 45 hours the signing process was finished.The memory usage while signing the zone maxed out around 9 GB.[13]

[12]The .com zone is growing by one million new domains each month.

[13]The writing of the signed zone file (**de..signed**) took about 9 hours. After which it took another 11 hours for the process to die. We do not precisely know what the signer was doing

7.1.3 Extrapolating the results

From this experiment we can calculate the maximum number of domains in a zone we can sign with DSA 768 keys on *this machine*. We needed 9 GB virtual memory for 16 M domains. That is about 562.5 bytes of memory per domain. We have a total of 20.256GB memory, so in theory the largest zone we could sign would be: $\frac{(20.256 \cdot 10^9)}{562.5} = 36,010,667$ This means we could do a delegation only zone file consisting of over 36M domains. We could sign the .com zone of 2001 on this hardware in a reasonable amount of time. When we made this information public everybody was stunned, including the InterNIC. Two weeks later we were in the possession of the real .com, .net, .org, .gov and the .in-addr zones. A full description of this experiment can be found in [13] and a description of the signing of .com in [15].

7.2 Signing Speeds

In these tests we measured the speed differences in the signing process. We used the .in-addr zone we had got from InterNIC. That zone consisted of 376,654 delegations and 518 PTR records. We used BIND9.0.0b4 on the same alpha.

We found that:

- the actual signing of a zone consumes less than half of the total time.
- in the first 10 minutes most of the time is spend looking for **.signedkey** files. If BIND9 does not find such a file, it will sign that zone with a NULL key, so you will need a .signedkey file for every delegation to a secure zone in the master zone. For the .com domain this means that they will ultimately have a directory filled with 16 million .signedkey files. Clearly this is not an options, even if it is possible to fill a directory with 16 million files, every directory readout will just take too much time. A signedkey database could be a solution to this problem.
- there is little or no difference in the memory consumption plots of DSA-signing and RSA-signing. The duration of the process is much longer with RSA. With the same key length RSA is a factor 15-20 slower than DSA.[14]

The following tests were run:

key length	time to complete signing (min)
DSA-768	16
DSA-1024	25
RSA-768	250
RSA-1024	500

The memory has a linear relation with respect to the size of the zone. BIND9 is very efficient in signing a zone. The process also looks CPU bounded. More

in that time; probably the OS was reclaiming the freed memory.

[14] This is solved with the final version of BIND.

power would certainly speed up the process. The complete report of these tests can be found in [14].

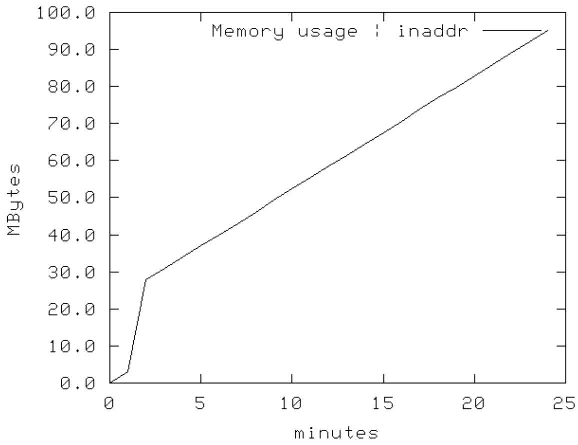

Figure 15: The memory consumption while signing with DSA-1024.

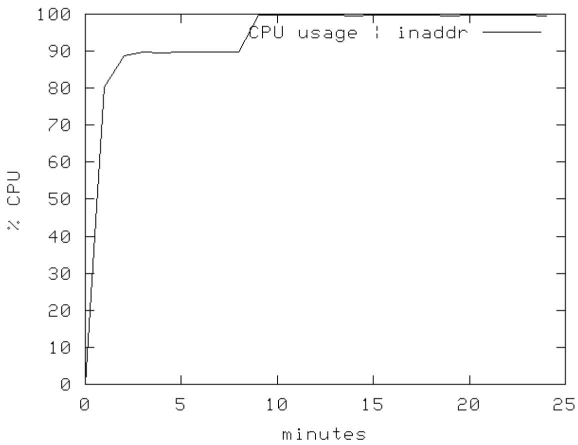

Figure 16: The CPU usage while signing with DSA-1024.

8 Implementing DNSSEC

8.1 Why BIND

When it comes to DNS there aren't too many other packages available besides BIND. Now with DNSSEC this is even more true than before, the alternative DNS implementation that are available haven't even began to implement the DNSSEC protocol. Further more BIND9 code is open source and therefore freely available with a zero price tag. BIND has proven itself for a long time know and it likely to remain the standard DNS(SEC) software for some time to come.

8.2 Alternative DNS implementations

There are other DNS implementations, like the Windows NT Domain Controller from Microsoft, NUBIND and djbdns. The Microsoft implementation only runs on Windows and has compatibility problems with BIND. NUBIND is a patched version of BIND8 with non standardized support for Unicode, to support for instance domain names that are typeset in the Korean alphabet. This internationalization of the domains names creates it own set of problems, both political and technical also see [20]. Djbdns is written by D. J. Berstein (http://cr.yp.to) and is used by some sites.

8.3 Case study .nl.nl

The zone .nl.nl is set up to start a deployment of DNSSEC in the Netherlands. The purpose is to mirror a large portion of the regular .nl tree in a secure fashion. For this to happen a secure registry must be implemented, which delegates the .nl.nl domains and is a secure entry point. This also means taking care of a lot of administrative issues like giving out new .nl.nl domains, just like the actual registry (http://www.nic.nl). The goal is to implement this secure registry in such a way that everything is automated.

8.4 Design

The secure aware registry will be designed according to the following guidelines:

1. although it is not a standard, .nl.nl will go forward with the idea to place the parent's sig over the child's key in the parent's zone. The belief at NLnet Labs is that this will make the administrative side of DNSSEC a lot easier.

2. the secure registry must also be very similar to the standard .nl registry (SIDN). This will mean that a lot of the same functionality must be implemented.

3. the interface for registry will be based on (mail)forms that can be processed automatically.

4. only domainholders with a domain under .nl can apply. See paragraph 8.5.1 on why this is a good idea.

It is expected that this registry should be finished in the second half of 2001. There should also be a rudimentary version of a secure aware resolver by that time. Together these form a very good proof of concept of RFC 2523 [11].

8.5 Current setup in .nl

The current registry in the Netherlands has a solid technical and administrative organization. There are two ways to interface with the system.

1. sending email forms to hostmaster@nic.nl.
2. downloading java/oracle forms for use inside a browser.

Both interfaces essentially use the same forms, I will list the more important ones below:

request a new domain by sending in this form, one can become the owner of a domain in the .nl zone.

move a domain to a new holder with this form, one can sell a domain to someone else.

change data concerning a domain used to for instance change a typo in the name of the holder. There are multiple forms to change different things

status of a request in the Netherlands not all domain names are allowed. With this form one can check if the name is accepted or not

And with `whois` it is possible to check if a domain name is already taken or not.

The following extra steps are needed in DNSSEC, they are all related to security and keys.

security request first step in getting secure. The steps in figure 10 are executed.

new key request a child zone want to use a new key. See figure 13.

new signature request in our setup the parent (the registry) decides when and if a child gets a new signature.

The .nl.nl registry must at least implement the above requests and forms, with a possible exception for a status form, as only .nl names can be made secure.

8.5.1 Problems encountered

The choice for .nl.nl as a test bed gave some unusual problems. When we had .nl.nl running at one of our servers we immediately began seeing queries for domains in the .nl.nl zone. This is strange because there were no domains yet in the .nl.nl zone. After further investigating we came to the conclusion that there were some nameserver who just appended .nl to all the zones they were trying to resolve and thus were delegated to our server.

As long as .nl.nl doesn't actually delegate zones it will give a NXDOMAIN back on all those queries. The resolvers would then try it again, but without the extra .nl. To test how much of a problem this could get, we made a wildcard delegation for *.nl.nl and got a website running on www.nl.nl, or in DNS speak: `*.nl.nl.  IN CNAME www.nl.nl.`

In a matter of hours we had approximately 150,000 hits on our website and people began to email us why they couldn't use the Internet.[15] We've shut down the website and deleted the wildcard delegation for .nl.nl that same evening. Although the TTL for .nl.nl was very low (2 hours) some people were still experiencing problems three days after the experiment. It seems that our web page was cached somewhere on an important proxy server and when people requested a page on the Internet this request was processed by this proxy and it then served up our page.

To prevent this from happening again when .nl.nl is delegating domains it is important that the delegation points to the same nameservers as the corresponding .nl zone does. In other words .nl.nl must be an exact mirror of .nl. This gives another advantage when .nl.nl is as big as .nl – at this point the whole .nl is securely mirrored in .nl.nl – we only have to strip a .nl of all the names and the entire secure tree is copied over to .nl making the .nl tree of the Netherlands instantly secure.

8.6 Outstanding issues

These issues .nl.nl is trying to solve.

key duration there is little or no knowledge about an optimum lifetime of a key. The key should not be used too long to avoid the possibility of cracking the private key. On the other hand you want to use a key as long as possible to avoid resigning a large zone or a least minimize the effort.

hands on experience This is one of the reasons to begin with .nl.nl in the first place. This is the first experiment with DNSSEC by any registry on this scale.

8.6.1 Legal issues

The .nl.nl registry is a test bed with no security guarantees what so ever, so there is no need for legal contracts. When DNSSEC is about to be implemented

[15] Most of these people were from one big Dutch provider.

these contracts must of course exist. It is important to see that although it looks like the parent zone gets more control over a child zone, this is not the case. Consider a bad delegation of the parent in the current DNS. This will mean that a child zone is not reachable on the Internet. With DNSSEC there are *more* possibilities to create this situation but that doesn't mean a parent has *more* power over a child zone. If there is a parent screw up in DNSSEC the worst thing that can happen is that the child becomes bad and drops of the Internet. Thus there is no difference in the parent's power over the child between DNS and DNSSEC. It is however a lot easier to create a bad zone in DNSSEC (just wait for the signature to expire) than it is to create a lame delegation in DNS (there must be made a change somewhere).

8.6.2 Modifications to BIND

I have tested the impact of our proposal for the location of key sig on BIND9. The test consisted of just setting up a secure child zone under .nl.nl and to check if BIND9 could properly handle this. This proved to be no problem at all. This behavior of BIND9 was later confirmed by Brian Wellington (co-writer of BIND9 and creator of xbill). This means that for .nl.nl *no* changes are needed in the existing codebase of BIND9.

9 Conclusions

When I started this master thesis it was expected that only administrative issues stood in the way of a widespread deployment of DNSSEC. Now with a lot of (political) developments in the DNSSEC community, like a possible complete rewrite of RFC 2535, the deployment could be moved back a further 14 months. As this master thesis provides a number of possible solutions for the problems of DNSSEC, this does not have to be the case.

With DNSSEC the size of zone files will increase dramatically, but this will pose not much of a problem. The .com registry has already shown the rest of the world how to handle big zones.

To use DNSSEC, zones need to be signed. It was thought this would not be possible for big zones, like .com, .de or even .nl. We have shown that even zones as big as .com can be signed. For .com this took 50 hours. This was done on a single machine with only one (64 bit) CPU. It is expected that with bigger hardware this time can come down to a level on which the .com zone can be signed twice a day.

One of the biggest problems with DNSSEC is how to manage all the keys and signatures it requires. This is actually a very old problem (PGP also creates such a problem), but now with DNSSEC really it needs to be solved. The procedures presented in this master thesis provide a possible solution. Also, signatures for zones must be recreated on a regular basis, every month an update is normal in DNSSEC. This will require a lot of extra communication between a parent and a child zone. If the signature the parent creates over the child's key is stored in the parent's zone file, this communication can be minimized. This will break a basic DNS assumption that all the data of a zone must be located in the authoritative nameservers of that zone. It could be possible to make an exception for this case. It is however up to the DNS/DNSSEC community to decide what they want. It must be noted, that this master thesis clearly states that right now – with the parent's sig over child's key at the child's zone – DNSSEC will be *very* difficult, if not impossible, to administer when dealing with large zones.

With the problems found in DNSSEC and the solutions given in this master thesis, NLnet Labs has decided to begin writing a secure aware resolver for use in DNSSEC, something that is *still* lacking right now. Without this resolver DNSSEC is only half finished. This resolver will take into account that the parent's sig (over the child's key) is stored in the parent's zone. My future work at NLnet Labs will consist of building the first secure aware registry. When these two project are successfully completed .nl.nl will be the first (big) secure domain in the world and completely comply with RFC 2535 and a resolver which end users can use. It is expected that this work will take 6 months. This will mean that DNSSEC could be deployed in the Netherlands in the second half of 2001. That is 12 months earlier than anticipated by the DNSEXT work group of the IETF.

Acknowledgments

First and foremost, I'd like to thank my supervisors of this master thesis project: Ted Lindgreen of NLnet Labs in Amsterdam and Bart Jacobs of the University of Nijmegen. Other people that gave valuable input are Roy Arends, Jaap Akkerhuis, Marcos Sanz and Olaf Kolkman.

I would also like to thank Leo Willems of Tunix how gave me the opportunity to get this master thesis assignment at NLnet Labs and Caspar Terheggen for giving me some extra insight in DNSSEC.

During the making of this master thesis I've had the opportunity to go to Sweden, to attend a DNSSEC workshop. I've met a lot of people there and it was very interesting to see how the development and deployment of something like DNSSEC is setup. Thanks to NLnet Labs I've also been to an IETF meeting in San Diego, which was also great.

I will stay involved with the development of DNSSEC and will continue to work for NLnet Labs for at least one year. In that time I will also try to finish my study computer science in Nijmegen.

10 List of Acronyms

acronym	description, (first occurrence)
admin-c	Administrative Contact, (25)
BIND	Berkeley Internet Name Domain, (6)
ccTLD	Country Code Top Level Domain, (20)
DNS	Domain Name System, (6)
DNSEXT	Domain Name System EXTentions, (34)
DNSSEC	Domain Name System SECure, (13)
DoS	Denial of Service, (3)
DSA	Digital Signature Algorithm, (15)
gTLD	Generic Top Level Domain, (20)
IANA	Internet Assigned Numbers Authority, (16)
IETF	Internet Engineering Task Force, (34)
IP	Internet Protocol, (6)
IPSEC	Internet Protocol SECure, (15)
ISP	Internet Service Provider, (22)
LAN	Local Area Network, (6)
NIC	Network Information Center, (6)
PGP	Pretty Good Privacy, (16)
PKI	Public Key Infrastructure, (27)
RFC	Request For Comments, (4)
RR	Resource Record, (9)
RSA	Rivest Adi Adleman, (15
SIDN	Stichting Internet Domeinregistratie Nederland, (22)
SOA	Start Of Authority, (9)
tech-c	Technical Contact, (25)
TLD	Top Level Domain, (10)
URI	Uniform Resource Identifier, (16)

References

[1] Steven M. Bellovin. *Using the Domain Name System for System Break-ins.*
AT&T Bell Laboratories, 1995

[2] Paul Albitz and Cricket Liu. *DNS and BIND.*
O'Reilly & Associates, Sebastopol,1992

[3] Craig Partridge *Mail routing and the domain system.*
Request for Comments 974, Internet Engineering Task Force, January 1986.

[4] Paul Vixie. *A Paper regarding DNS and BIND Security Issues*
Originally published in the proceedings of the 5th Usenix Security

Symposium,
http://www.usenix.org/publications/library/proceedings/security95/full_papers/vixie.txt

[5] M. Lottor. *Domain administrators operations guide.*
Request for Comments 1033, Internet Engineering Task Force, November 1987.

[6] P. Mockapetris. *Domain names – concepts and facilities.*
Request for Comments 1034, Internet Engineering Task Force, November 1987. Obsoletes RFC0973; Updated by RFC1101

[7] P. Mockapetris. *Domain names – implementation and specification.*
Request for Comments 1035, Internet Engineering Task Force, November 1987. Obsoletes RFC0973; Updated by RFC1348

[8] (Dutch) Anne-Wil Duthler. *Met recht een TTP.*
Kluwer, Deventer, 1998

[9] M. Andrews. *Negative Caching of DNS Queries (DNS NCACHE).*
Request for Comments 2308, Internet Engineering Task Force, March 1998. Updates RFC1034 and RFC1035

[10] Y. Rekhter, B. Moskowitz, D. Karrenberg & G. de Groot *Address Allocation for Private Internets.*
Request for Comments 1597, Internet Engineering Task Force, March 1994.

[11] D. Eastlake *Domain Name System Security Extensions.*
Request for Comments 2535, Internet Engineering Task Force, March 1999. Obsoletes RFC2065; Updates RFC2181, RFC1035, RFC1034

[12] R. Elz & R. Bush *Clarifications to the DNS Specification.*
Request for Comments 2181, Internet Engineering Task Force, July 1997. Updates RFC1034, RFC1035, RFC1123

[13] R. Arends *Signing the German TLD zone.*
July 2000. http://www.nlnetlabs.nl/dnssec/de-signing.txt

[14] R. Gieben *Signing of really big zones.*
September 2000. http://www.nlnetlabs.nl/dnssec/bigzones.html

[15] R. Gieben *Signing of .com.*
October 2000. http://www.nlnetlabs.nl/dnssec/sign.com.html

[16] B. Sahlin *New challenges to the Domain Name System: extension for security, dynamic updates and IPv6.*
November 1999. http://www.tml.hut.fi/~bos/dns.html

[17] A. Romao *Tools for DNS debugging*
Request for Comments 1713, FYI 27, November 1994

[18] O. Gudmundsson *Storing Certificates in the Domain Name System (DNS)*
Request for Comments 2538, Internet Engineering Task Force, March 1999

[19] B. Schneier *Applied Cryptography: Protocols, Algorithms, and Source Code in C.*
John Wiley & Sons, 1995

[20] (Dutch) J. Akkerhuis *Status van Internationale domeinnamen.*
September 2000. http://www.nlnetlabs.nl/idn/index.html

[21] R. Needham and M. Schroeder *Using Encryption for Authentication in Large Networks of Computers*
1978. Communications of the ACM, 21, pp.393-399

[22] Man & Mice *Domain Health Survey*
August 2000. http://www.menandmice.com/infobase/mennmys/vefsidur.nsf/index/6.1.2000.8

[23] M. Bellare and P. Rogaway *Entity Authentication and Key Distribution*
August 1993. Advances in Cryptology - Crypto '93 Proceedings

[24] M. Andrews and D. Eastlake *Domain Name System (DNS) Security Key Rollover* M. Andrews and D. E. Eastlake 3rd
Internet draft. Updates RFC1996. Expires January 2001. Internet Engineering Task Force July 2000. http://search.ietf.org/internet-drafts/draft-ietf-dnsop-rollover-00.txt

[25] S. Josefsson *Authenticating denial of existence in DNS with minimum disclosure* Internet draft. Expires May 2001. Internet Engineering Task Force. November 2000. http://search.ietf.org/internet-drafts/draft-ietf-dnsext-not-existing-rr-01.txt

[26] R. Clarke *Conventional Public Key Infrastructure: An Artefact Ill-Fitted to the Needs of the Information Society* Euro. Conf. in Inf. Syst. (ECIS 2001), November 2000. http://www.anu.edu.au/people/Roger.Clarke/II/PKIMisFit.html

[27] O. Gudmundsson *DNSSEC and IPv6 A6 aware server/resolver message size requirements* Internet draft. Expires March 20001. Updates RFC2535 and RFC 2874, Internet Engineering Task Force. October 2000. http://www.ietf.org/internet-drafts/draft-ietf-dnsext-message-size-01.txt

[28] R. Gieben, T. Lindgreen *Parent's SIG over child's KEY* Internet draft. Internet Engineering Task Force. January 2001. http://www.ietf.org/internet-drafts/draft-ietf-dnsop-parent-sig-00.txt