UNIVERSITY OF TWENTE.

Master Thesis

The Performance of ECC Algorithms in DNSSEC: A Model-based Approach

Faculty: Electrical Engineering, Mathematics and Computer Science
Group: Design and Analysis of Communication Systems

Author
Kaspar Hageman
University of Twente
k.d.hageman@student.utwente.nl

Committee
Roland van Rijswijk-Deij, M.Sc
prof. dr. ir. Aiko Pras
dr. Anna Sperotto
dr. Ricardo de O. Schmidt

October 21, 2015

Abstract

The Domain Name System (DNS) resolves domain names to IP addresses on the Internet. Several vulnerabilities in DNS led to the development and deployment of a secure extension, DNSSEC, which provides authentication and data integrity by including digital signatures in DNS responses. DNSSEC has its own flaws however, of which DDoS amplification potential, fragmentation related issues and deployment complexity are the most severe ones. Where the RSA signature scheme is currently widely deployed in DNSSEC, its elliptic curve (ECC) alternative, Elliptic Curve Digital Signature Algorithm (ECDSA), is more recently standardized and supposedly reduces the aforementioned flaws, by reducing the size of digital signatures and cryptographic keys. EdDSA is another, more recent, digital signature scheme based on elliptic curve cryptography, and has even more promising properties (e.g. smaller key size). The major drawback of ECC is that the validation of signatures is computationally more intensive than RSA. A transition from RSA towards ECC would introduce a significant increase in computational load caused by signature validations for DNS resolvers. While some simple benchmark tests confirm that a single ECC validation is computational costly, there exists no scientific proof that ECC can be deployed on a large scale without causing any performance issues. In our research we have developed a DNSSEC model based on measurement from three deployed resolvers. The resulting regression model was applied to evaluate several scenarios, both current and future scenarios. Based on the scenarios, we can conclude that the switch towards ECC can be made without encountering any computational related issues for validating resolvers.

This research was conducted in collaboration with SURFnet, the Dutch national research and education network which provides Internet and services to the research and education community in the Netherlands.

Contents

List of terms

Abbreviations

AAF	Amplification Attack Factor
CA	Certificate Authority
CDF	Cumulative Distribution Function
CHR	Cache Hit Ratio
CSK	Combined Signing Key
DANE	DNS-based Authentication of Named Entities
DDoS	Distributed Denial of Service
DNSSEC	Domain Name System Security Extensions
DNS	Domain Name System
DSA	Digital Signature Algorithm
DoS	Denial of Service
ECC	Elliptic Curve Cryptography
ECDF	Empirical Cumulative Distribution Function
ECDSA	Elliptic Curve Digital Signature Algorithm
EDNS0	Extension Mechanisms for DNS
EdDSA	Edwards Curve Digital Signature Algorithm
Eemo	Extensible Ethernet MOnitor
FQDN	Full Qualified Domain Name
ICANN	Internet Corporation for Assigned Number and Names
ISP	Internet Service Provider
KSK	Key Signing Key
MTU	Maximum Transmission Unit

MUX	Multiplexer
NIST	National Institute of Standards and Technology
NS	Name Server
RR	Resource record
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm
SLD	Second-Level Domain
SLR	Simple Linear Regression
TCP	Transmission Control Protocol
TLD	Top-Level Domain
TTL	Time To Live
UDP	User Datagram Protocol
ZSK	Zone Signing Key
ccTLD	Country Code Top-Level Domain
eBATS	ECRYPT Benchmarking of Asymmetric Systems
gTLD	General Top-Level Domain

Parameters

α_s	The fraction of responses containing signatures
α_v	The fraction of signatures being validated
$\bar{r}$	The average number of responses per outgoing query
$\bar{s}$	The average number of signatures per signed response

Variables

Q	Outgoing queries per second
R	Incoming responses per second
R_n	Incoming responses without signatures per second
R_s	Incoming responses with signatures per second
S	Incoming signatures per second
S_n	Incoming non-validated signatures per second
S_v	Signature validations per second

Chapter 1

Introduction

The Domain Name System (DNS) translates human-readable host names (e.g. `www.example.com`) to IP addresses (e.g. `93.184.216.34`). This translation occurs at *name servers* to which a client sends its queries, asking for the IP address of a particular host name. Intermediate systems called *recursive DNS resolvers* often perform DNS lookups on behalf of many clients. To reduce the network traffic load and latency, DNS resolvers store retrieved DNS answers in a local cache for answering future identical requests.

At the time of DNS development, security of the system was not taken into account. In 1993, the first vulnerabilities in DNS were reported, but a very severe one was discovered in 2008: the Kaminsky attack. In the Kaminsky attack, the most severe form of "cache poisoning", an attacker targets a DNS resolver by inserting malicious data in the cache of the resolver. An attacker triggers a DNS lookup at the target and tries to send a spoofed response that looks like it originated from the authoritative name server (i.e. the name server responsible for the requested domain name). When successful, an attacker can redirect legitimate clients of that particular resolver to an address of his choice. Cache poisoning attacks are possible because a DNS resolver is not able to verify the authenticity and data integrity of the received response.

As a response to the existing DNS vulnerabilities, the Domain Name System Security Extensions (DNSSEC) protocol was designed. DNSSEC provides the authenticity and data integrity that DNS lacks. It achieves this by attaching a digital signature to each DNS response, which proves that only the claimed sender could have sent the message and that the data was not tampered with. Signatures are generated with public-key cryptography schemes, such as RSA or ECDSA, where the private key is used to create a signature and the public key is used for validation. Establishing a chain of trust provides complete authenticity and data integrity. Usually, name servers use two cryptographic keys as part of the chain of trust: the Zone Signing Key (ZSK) is used to sign the DNS data, where the Key Signing Key (KSK) is used to sign both keys. Attaching signatures and keys to DNS responses causes them to become much larger than traditional DNS responses.

While DNSSEC provides protection from cache poisoning attacks, it introduced several new issues. Firstly, the protocol can be easily abused in so-called amplification attacks, a subset of Distributed Denial of Service (DDoS) attacks. Amplification attacks rely on the fact that DNS responses are much larger than the requests, which allows an attack to generate large traffic volumes by only sending a relatively small amount of traffic. Although DNS already suffered from amplifi-

cation attacks, the large DNSSEC keys and signatures have increased this problem. Secondly, the large keys and signatures cause DNSSEC responses to exceed the Maximum Transmission Unit (MTU) more often, which causes these packets to be fragmented. Misconfigured firewalls refuse to forward fragmented packets, making clients and resolvers behind the firewall unable to resolve DNS queries. The choice of signature algorithm could potentially reduce these two issues.

RSA is currently used in most DNSSEC deployments to create digital signatures. More recently its elliptic curve cryptography (ECC) alternative, Elliptic Curve Digital Signature Algorithm (ECDSA), was standardized in DNSSEC [1] and this algorithm has some benefits over RSA. Edwards-curve Digital Signature Algorithm (EdDSA) is a non-standardized algorithm, but has even more favorable properties than ECDSA [2]. RSA and ECC provide the same cryptographic functionality in DNSSEC, but due to fundamental differences in the underlying mathematics they have very different properties. The most notable of these are as follows:

Key size - ECDSA has a much smaller key length than RSA [3, 4]. The key size of EdDSA is even smaller.

Cryptographic strength - RSA - considering its most popular version with 2048 bits - is cryptographically weaker than the weakest standardized ECDSA version, P-256 [5], as well as EdDSA.

Signature size - The size of P-256 ECDSA and EdDSA signatures is significantly smaller than 2048-bit RSA [3, 4].

Computation cost of signature validation - Validation of an ECDSA signature takes much more time than validation of an RSA signature. EdDSA performs better than ECDSA, but is still slower than RSA.

The smaller key- and signature size of ECDSA and EdDSA causes DNSSEC responses to become smaller. This would reduce the two aforementioned issues with DNSSEC: DDoS potential and fragmentation issues. In addition, ECC may allow us to replace the commonly used ZSK/KSK scheme with a Combined Signing Key (CSK) scheme, which would reduce the size of DNSSEC responses even more. The potential widespread deployment of ECC is met with reluctance, because the impact on the computation load of validating resolvers is unknown. There currently exists no extensive research into the performance of ECC in DNSSEC. Additionally, the impact of ECC deployment on the DDoS potential and fragmentation issues of DNSSEC is also not known.

1.1 Research goal

In our research we have answered the following question: *"What is the effect of deploying ECC as a replacement of RSA in DNSSEC on the computation load of validating DNS resolvers?"*. The answer was obtained by predicting the number of signature validations in different scenarios, using a prediction model. The model was developed based on passive measurements on the DNS traffic of a set of deployed DNS resolvers. The data set was supplied by SURFnet, the National Research and Education Network in the Netherlands[1]. Given the number of outgoing queries per second, the linear model predicts the number of signature validations per second. The model was validated with different data sets to prove that the model is indeed a good predictor of the number of signature validations. Several scenarios (both current and future scenarios) were defined in order to answer the research question. The results show that even in the most unfavourable conditions, a validating resolver is able to cope with the computation load caused by

[1] http://www.surfnet.nl/en/

potentially deploying ECC globally as signature scheme in DNSSEC. Additionally, the DNSSEC infrastructure is sufficiently future-proof because even with a full DNSSEC deployment (i.e. every single domain in the world is signed), the computation load from signature validations is still no issue.

1.2 Thesis structure

The remainder of this thesis is structured as follows. First, Chapter 2 provides a background of DNS which includes the relevant details of DNS(SEC) and digital signatures. The background chapter provides the required knowledge for those unfamiliar with DNS. Secondly, the problem of the research is specified in several research questions in Chapter 3. This followed by Chapter 4 in which the related work of the research is described. Chapter 5 states the approach that was taken to answer the research questions. The results of the used approach are described in Chapters 6, 7 and 8. The thesis is ended with a discussion in Chapter 9 and the conclusions in Chapter 10.

Chapter 2

Background

2.1 DNS

The Internet works by allocating a unique IP address to every endpoint, such as hosts, servers or routers. IP addresses are labels that consist of 32 bits in case of IPv4 or 128 bits in case of IPv6. Resources on the Internet are often referred to by their names, rather than their IP addresses. For example, instead of accessing `93.184.216.34` the domain name `www.example.com` can be accessed to retrieve a web page. For us humans it is much easier to remember domain names instead of IP addresses. For each host name on the Internet, there is a mapping to the corresponding IP address. With millions of endpoints on the Internet, it is infeasible to maintain a table with this mapping (which was used before DNS). The Domain Name System protocol solves this issue by translating the hostnames to the IP address in a hierarchical and distributed fashion.

2.1.1 Domain name space

DNS uses a hierarchical (tree) structure. The top of this structure is called the root, and is denoted by a dot (`.`). The second level of the hierarchy consists of Top-Level domains (TLDs), followed by Second-Level domains (SLDs) and zero or multiple lower levels. The TLDs can be separated into two groups: Generic Top-Level domains (gTLDs), e.g. `com`, `org` and `net`, and Country Code Top-Level Domains (ccTLDs), e.g. `us`, `nl` and `uk`. Each of the hierarchy layers is separated by a dot. Each domain name in DNS consists of one or more parts, named labels. The rightmost label of a domain name conveys the highest domain in the DNS tree structure. Figure 2.1 shows how the domain name `www.example.com.` represents its place in the DNS tree hierarchy. The domain name can be read from right to left, where the '`.`' is the root, `com` is the gTLD, `example` is the SLD and `www` is the hostname. In this example the domain name includes the root symbol, but this is often discarded. A domain name that includes the root is called a *Full Qualified Domain Name (FQDN)*.

2.1.2 Domain authority and delegation

Each node in the domain name space is assigned to an authority, i.e. a party that is responsible for the management of that node. Additionally, the party is also responsible for the entire sub tree under the particular node. For example, the root node of DNS is authorized by the Internet Corporation for Assigned Number and Names (ICANN). The authority of a node can *delegate* authority

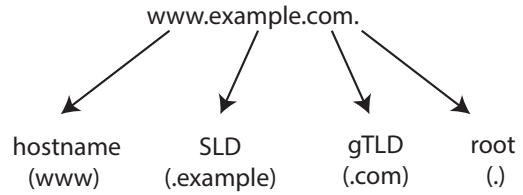


Figure 2.1: The hierarchy of the DNS can be read from right to left, where the dot denotes the root and `www` denotes the hostname.

for lower levels of that node to other parties. The part of the domain space that is authorized by a single party is referred to as a *zone*. All authorities maintain one or multiple name servers, which are able to respond to DNS requests for its zone. A name server that delegates authority to other parties should be able to refer to a lower level name server that could answer the query.

Authoritative name servers maintain information of their zones in so called *zone files*. These zone files consist of a collection of resource records (RRs). Each record has a name, a type (e.g. A, NS, MX, etc.), a time-to-live (TTL), a class and type specific data. For example, an `A` record contains an IPv4 address, an `AAAA` record contains an IPv6 address and an `NS` record contains the name of an authoritative name server. Multiple resource records that share the same name, type and class are referred to as a *resource record set* (RR set). Essentially, the DNS RR sets form one giant, distributed database.

Resource records can be retrieved by sending DNS queries to name servers, which will respond with the associated RR set, will send an `NXDOMAIN` response (i.e. the requested record does not exist) or will refer the requester to a lower level name server that may be capable of providing the answer.

2.1.3 DNS queries

All DNS communication is carried in a single message. The DNS message consists (besides a header) of (i) a question section, (ii) an answer section, (iii) an authority section and (iv) an additional section. The question section contains the requested domain name. The answer, authority and additional sections contain a list of RRs. The answer section contains the resource records that answer the question section. The authority section points to the authoritative name server for the requested domain name. The additional section can contain supportive RRs, such as the IP addresses for authoritative name servers for the requested domain.

Figure 2.2 shows the workflow for a DNS query. The DNS query is invoked when a user wishes to resolve a hostname (such as `www.example.com` via a web browser). Note that in this example the browser invokes the DNS protocol, but other types of applications exist that use DNS as well. The following steps take place:

1. The browser invokes a local application named the *stub-resolver*, which is located on the user's computer system itself. The stub-resolver handles all DNS queries on behalf of the user.
2. The stub-resolver sends the query to a *recursive DNS resolver*, which is often operated by the Internet Service Provider (ISP).

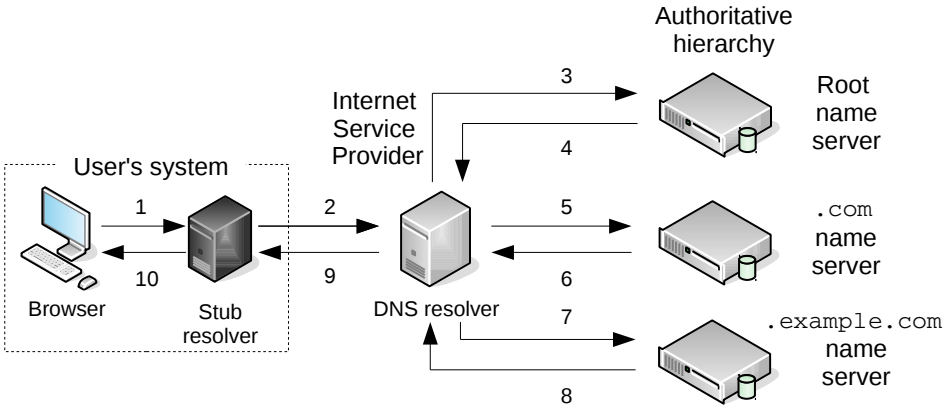


Figure 2.2: An example of a DNS query.

3. The DNS resolver sends the query to one of the authoritative name servers (NS) for the root zone. The resolver can reach the root servers, because the IP addresses of all root servers are static and are by default included in DNS software.
4. The root server is not authoritative for the domain `www.example.com`, but has the IP addresses of the authoritative NSs for the `com` domain. The root server responds with a 'referral' to all lower level NSs.
5. The DNS resolver sends the query to one of the `com` name servers.
6. The `com` NS is not authoritative for the domain `www.example.com`, but has the IP addresses of the authoritative NSs for the `example.com` domain. The `com` server responds with a 'referral' to all lower level NSs.
7. The DNS resolver sends the query to one of the `example.com` name servers.
8. The `example.com` NS is authoritative for `www.example.com` (i.e. its zone file contains an 'A' record for the domain name) and responds with `IP address = 93.184.216.34`.
9. The DNS resolver responds to the stub-resolver with the IP address.
10. The stub resolver responds to the browser with the IP address.

Since DNS is intended to be lightweight, the User Datagram Protocol (UDP) is widely adopted as transport layer protocol. DNS queries are sent from an arbitrary port to port 53. The associated responses have the reverse port numbers; the source port is 53, where the destination port is the arbitrary port that was defined by the requester [6]. While UDP is mainly used, the Transmission Control Protocol (TCP) is also supported [7]. Initially the maximum size of a DNS message was 512 bytes, although this limited size was increased later with the Extension Mechanism for DNS (EDNS0) [8].

For resolving a single domain name, DNS seems to generate a relatively large amount of network traffic and also seems to respond slow due to the number of messages exchanged. To solve these two problems, each DNS resolver maintains a cache. Each received RR set is stored in the cache until the *time-to-live (TTL)* expires. The TTL guarantees that cache entries will disappear eventually, so that RR sets need to be retrieved every once in a while. This ensures that changes in zone files eventually propagate to DNS resolvers. Whenever a DNS resolver

receives a DNS lookup, it first checks whether the requested domain name is cached. If so, the corresponding cached RR set is returned, instead of invoking another name server. The TTL value is determined by the zone operator and may vary per resource record.

2.1.4 Cache poisoning

One of the most severe threats to DNS is cache poisoning, and in particular the Kaminsky attack. As mentioned before, DNS resolvers maintain a cache of DNS records to minimize network traffic and to enhance the performance. In cache poisoning, a malicious party inserts so called bogus data in the cache of a DNS resolver, causing it to give the malicious data to querying clients. For example, an attacker might want to redirect clients of the domain name `www.examplebank.com` to his own web server, where he can set up a website that tricks clients into giving important data, while the clients themselves believe that they are browsing a legitimate website. The Kaminsky attack - named after its discoverer Dan Kaminsky - was discovered in 2008 and is the most harmful of the various variants of cache poisoning attacks that exist. It allows an attacker to compromise an arbitrary domain with a high success rate in a short amount of time [9].

The Kaminsky attack is initiated by the attacker by sending a DNS lookup request to the victim resolver. The domain name should not be cached by the resolver and the domain name should be within the domain that the attacker wishes to compromise. For example, an attacker queries the domain `notexistingdomain.example.com` to compromise the domain `example.com`. Since the victim resolver has not cached the query it will forward the request to one of the name servers of `example.com`. At this point, the DNS resolver expects an answer from the name server that gives an IP address, an `NXDOMAIN`, or a referral to a lower level authoritative name server. The attacker now generates a set of spoofed response messages that seem to originate from the `example.com` name server. The spoofed messages contain a referral (the *additional* section in DNS message) to a name server that is under control of the attacker. The victim resolver will accept the first received valid response as an answer, so a cache poisoning attack can be considered a race between an authoritative name server and the attacker. If the victim resolver accepts one of the spoofed message as a valid response, the malicious name server referral is cached. Future clients that request domain names within the domain `example.com` are redirected to the attacker's name server.

DNS resolvers only accept data when it is sent as response to a pending query; a response that contains unexpected data is dropped by the DNS resolver. This means that an attacker needs to carefully craft a response that makes sense to the victim's system. In order to craft a legitimate response, an attacker should match the following fields in the DNS message:

- The response to a query should arrive at the same UDP port as it was sent from. If the source port is randomized correctly, there are roughly 2^{16} possible source ports available.
- DNS resolvers often have multiple pending queries. Responses are correlated to the query using the 'query ID' field in the DNS header. Therefore, the DNS response should have the same ID as the DNS query. The query ID field in the DNS header is 16 bits long, which results in $2^{16} = 65536$ number of unique possible query IDs.
- The DNS response contains the same *question* section as the sent query.
- The *authority* section should only contain names that are within the same domain as the question.
- The *additional* section should contain the same NSs defined in the *authority* section.

The attacker himself initiated the original DNS query and thus is fully aware of the *question* section. The *authority* section is easily obtainable, since this field can be extracted from an answer to a regular valid DNS response. The same holds for the *additional* section, although the attacker intends to change these RRs. This leaves only the UDP port and the query ID as unknown variables for the attacker, which need to be guessed correctly.

When port and query ID randomization is implemented, there are approximately $2^{16} \times 2^{16} = 2^{32} \approx 4.3$ billion combinations possible. Previous implementations of resolver software often used a fixed UDP port and fixed query ID, which made guessing much easier for the attacker. Even though an attack's success rate can be heavily reduced by correctly implementing randomization, an attacker can invoke an almost unlimited amount of attacks, because the Kaminsky attack is independent of domain name. If the attack on `notexistingdomain.example.com` has failed, the attack can be slightly modified to target `notexistingdomain1.example.com`. Both attacks would have resulted in compromising the domain `example.com`. Hubert [10] showed that an almost undetectable Kaminsky attack can be performed with a 50% likelihood of success in six weeks. Even with countermeasures, cache poisoning attacks still poses a huge threat to DNS.

Cache poisoning attacks show that DNS resolvers should be able to verify the identity of name server, i.e. message authentication, and to detect data tampering, i.e. data integrity.

2.2 DNSSEC

Domain Name System Security Extensions (DNSSEC) provides the much needed authentication and data integrity that the traditional DNS lacks, by attaching digital signatures to each DNS response. The digital signatures are created using a hash function and public-key cryptography (described in more detail in Section 2.3). The protocol was standardized in RFC 2535 [11] and updated in RFCs 4033-4035 [12, 13, 14]. DNSSEC introduces a set of new resource record types to DNS:

- `RRSIG` - is used to store the digital signatures in DNSSEC. If a zone is DNSSEC signed, a cryptographic signature is generated for each resource record set (RR set) in the zone and stored in the corresponding `RRSIG` record.
- `DNSKEY` - contains the public key of a cryptographic key set. Where the private key is kept hidden from the outside, the public key is available to anyone who requests it. In DNSSEC, most zones use two types of keys, the Zone Signing Key (ZSK) and the Key Signing Key (KSK). The ZSK signs all RR sets in the zone, where the KSK signs both keys. More details of this design choice are given in Section 2.2.1.
- `DS` - or delegation signer contains the hashed value of a KSK `DNSKEY` record. The `DS` record is stored in the parent zone only and is used to verify the authenticity of the KSK `DNSKEY` record of the particular zone. Essentially, it establishes a *chain of trust* between parent and child zones (Section 2.2.1).
- `NSEC(3)` - stands for Next SECure and these records are used to prove authenticated denial of existence. The records that have been specified so far only provide authentication and integrity for existing domain names. However, an attacker can still send valid *empty* responses, since there exist no signatures for empty answers. DNSSEC solves this issue by introducing the `NSEC` record. The `NSEC` record contains all existing record types for a particular domain name in the zone, thus each domain name in a zone has its own `NSEC` record. Each `NSEC` record also contains the name of the next record (in lexicographical

order), so the NSEC records form a linked list. However, this allows attackers to *zone walk* and extract all domain names from a zone. To prevent this, the NSEC3 record stores the digest of the domain name instead of the domain name itself [15].

DNS is distributed over thousands of systems worldwide. An overnight change towards DNSSEC would not be possible, so the protocol is designed backwards compatible with DNS and is being deployed gradually. The domain name space is currently in a hybrid situation where part of the DNS zones are signed. As of today, 43.8% of the nl zone is signed while only 0.44% of the com zone is signed. DNSSEC records are only included in DNS messages when both the sender and the receiver (e.g. an authoritative NS and a DNS resolver) of the message are *security-aware* (i.e. have DNSSEC implemented). Resolvers that verify the authenticity of name servers are referred to as *validating* DNS resolvers.

Besides providing cryptographic signatures to DNS, DNSSEC is also an enabling technology for other secure applications. For example, the IETF currently promotes the use of DNS-based Authentication of Named Entities (DANE). In TLS, a set of ±160 Certificate Authorities (CAs) is inherently trusted by clients (e.g. provided by browsers at installation of the software), similarly to the root servers in DNS. Recent attacks [16] show that when successful, an attacker can issue fraudulent certificates and as such can cause severe damage. To reduce the attack surface (i.e. all entry points for an attacker to launch an attack), DANE uses the DNSSEC infrastructure next to or instead of CAs to provide authenticity, by introducing a new resource record type, TLSA. DANE provides a second certificate validation channel next to the TLS certificate chain of trust. In order to issue a fraudulent certificate, an attacker should compromise a CA *and* the DNS root.

2.2.1 Chain of trust

As mentioned before, RR sets are signed in DNSSEC using cryptographic algorithms to prove their authenticity. Usually, security-aware authoritative name servers use a so called Zone Signing Key (ZSK) to create these signatures. When a validating resolver receives a DNSSEC response with an RR set and a signature, it can request the authoritative name server for the public key and check if the signature corresponds with the RR set. However, to provide full authentication, the validating resolver should also be able to verify the authenticity of the received ZSK public key. Therefore the ZSK has been signed with another key, the Key Signing Key (KSK). Again, the authenticity of the KSK should also be verified and thus the digest of the public KSK is placed in the parent zone as a DS record. The DS record in turn is signed by another key set and this *chain of trust* can be traced all the way up to the root zone.

As an example Figure 2.3 displays the chain of trust for www.surfnet.nl. The RR set of www is signed with the ZSK of the domain surfnet.nl (①). The authenticity of the key is proven by a signature that is generated using the KSK of the same domain (②). The digest of the KSK is stored as DS record the parent's zone (i.e. nl zone) (③). In turn, the authenticity of the DS record is proven by a signature created with the nl ZSK (④). The subsequent links in the chain of trust proof the authenticity of their previous link (⑤-⑧) in a similar fashion as the aforementioned steps. After step 8, the authenticity of the domain name can be traced back to the KSK of the root zone. Since the root has no parent zone, the authenticity of this key cannot be guaranteed using a DS record. Each validating DNS resolver should configure a '*trust anchor*', i.e. a copy of the digest of the root server's KSK. The trust anchor is used to verify the authenticity of the top of the chain of trust (⑨).

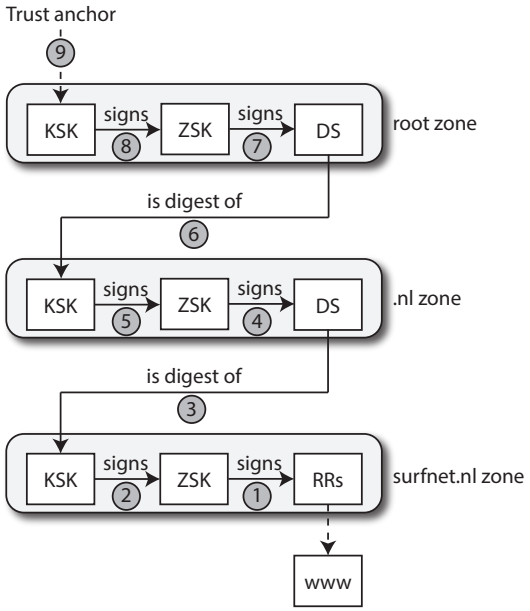


Figure 2.3: The chain of trust for `www.surfnet.nl`.

Re-signing and key rollover A regular practice in DNSSEC zone maintenance is zone re-signing and key rollovers. As mentioned before, each DNS RR set has a TTL, which specifies how long it should be cached by intermediate parties. Whereas the original DNS record types have a 'relative' TTL (i.e. the time that a record should be cached), the `RRSIG` type in DNSSEC has an 'absolute' TTL (i.e. an expiration time). After the expiration time of an `RRSIG` is passed, a signature should not be trusted anymore by a resolver and a new one should be retrieved. This implies that an authoritative name server should provide a new valid signature with an updated expiration time before the previous one expires, because otherwise the signed RR sets cannot be validated by any DNS resolver. Creating these new signatures is referred to as 're-signing the zone'.

In case of a key compromise, an authoritative party should be able to generate a new key set and resign the affected zone with the new key set. Additionally, to reduce the potential of cryptographic attacks, it is recommended to retire old key sets once in a while and employ a new one. Replacing an old key set with a new one is called a 'key rollover'. During a key rollover, the affected zone publishes multiple `DNSKEY` records simultaneously, such that cached RR sets can still be validated and newly signed RR sets can be validated using the new key set [17].

ZSK and KSK The 'chain of trust' example shows that DNSSEC employs two different key sets: the KSK and the ZSK. Introducing an extra set seems overly complicated and unnecessary. A situation where the ZSK is removed and the KSK is used to create signatures of RR sets (i.e. using a Combined Signing Key (CSK)) would function essentially the same [18]. Signatures and keys are preferred to be small (because this reduces network traffic and speeds up validation), and still provide a cryptographic strong system. Strong cryptography can be fulfilled by (*i.*) having a large key and signature size or (*ii.*) regularly performing a key rollover. Since the first option is infeasible with RSA, a key rollover should be performed often. In case of a CSK however, during a key rollover the parent's `DS` record should also change, and thus there is a parent-child

interaction required. A `DS` record update is performed outside the DNS protocol, and generally requires human intervention. It is undesirable to perform a `DS` update action often and therefore a KSK/ZSK mechanism is used. The larger KSK provides security and is rolled irregularly, where the smaller ZSK is used to sign RR sets and is rolled regularly.

2.3 Digital signatures

A digital signature is a method of demonstrating the authenticity and integrity of a message. Historically, digital signatures have been used in cases where forgery and tampering detection is important, such as financial transactions. All digital signatures schemes used in DNSSEC are based on hash functions and public-key cryptography. Both concepts work with so called *trapdoor functions* that are easy to perform in one direction, but extremely difficult to invert. Figure 2.4a shows a global overview of how a digital signature of an arbitrary message is created. The message (M) is first transformed using a one-way hash function. The output of that message (H(M)) is subsequently encrypted using an asymmetric cryptographic algorithm, with the signature as result (S(H(M))). The required validation steps are shown in Figure 2.4b; the receiver decrypts the signature (S(H(M))) and applies the hash function to the message. If both the received and calculated digests are equal, the validation is successful. In signature schemes, the private component of the key set is used to sign the message, where the public component is used to validate the message. This guarantees that only the owner of the private key can generate signatures, where everyone in possession of the public key is able to validate the message.

Hash functions are one-way functions that transform a message of arbitrary length into a fixed-length bit string, which is referred to as the digest. In DNSSEC, SHA-1 and the SHA-2 family hash function (Secure Hash Algorithm) are standardized in the required and recommended algorithms [19]. Hash functions are considered trapdoor functions because given input and the algorithm, it is very easy to calculate the digest, while given the digest and the used algorithm, it is very difficult to find the input. The details of SHA-1 and SHA-2 are beyond the scope of this thesis, but for more information take a look at the SHA standards publication[1].

In our research, we focus on three public-key algorithms: RSA, ECDSA and EdDSA. RSA is currently the most used algorithm in DNSSEC, where Elliptic Curve Digital Signature Algorithm (ECDSA) is standardized as the elliptic curve alternative and is rarely used. EdDSA - named after the twisted Edwards curves - is not adopted as standard in DNSSEC, but has promising properties and therefore we wish to include the algorithm. In the following sections a brief introduction of the algorithms is given and the relevant differences between the three are presented.

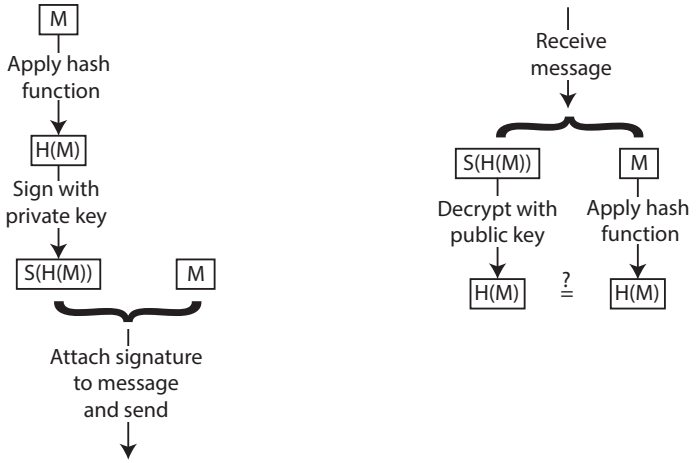
2.3.1 RSA

One of the most well known public-key algorithms is RSA; the name is derived from its inventors Ron Rivest, Adi Shamir and Leonard Adleman. RSA's security lies in the difficulty of factoring large numbers into primes. The public and private key are functions of large prime numbers [3, Chapter 19].

Key generation To generate a key pair, one should first choose two random prime numbers p and q. Compute the value of n:

$$n = pq$$

[1] http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf



(a) Signing of a message M.

(b) Validation of a received message.

Figure 2.4: General steps in the signing and validation phase.

Then (randomly) choose the public key e, such that e is relatively prime to *(p-1)(q-1)*. Then the private key d can be computed as follows:

$$ed \equiv 1 \bmod (p-1)(q-1)$$

At this point, primes p and q can be discarded. It is crucial that these primes are never revealed, since the private key can be derived from these numbers. The public key now consists of e and n, where d is the private key. To speed up RSA signing, the value of e is often chosen as 3, 17 or 65537 ($2^{16}+1$) [3, page 469].

Signing and validation Several signature schemes of RSA are described in PKCS#1 [20], where the RSASSA-PKCS1-V1_5 and EMSA-PKCS1-v1_5 specifications are used in DNSSEC [19]. Before signing the message with RSA, the message is encoded as a particular prefix and the digest of the original message. The prefix depends on the hash function used. After encoding the message, the signing of encoded message m into a cipher text c goes as follows:

$$c = \text{encoded}(m)^d \bmod n$$

Given the cipher text c and the public key (e, n) a receiver can recover the original message m as follows:

$$\text{encoded}(m) = c^e \bmod n$$

Removing the prefix from the encoded message m results in the digest of the original message. The strength of RSA lies in the fact that given the value of n, finding the prime factorization into p and q is extremely time consuming (at least assuming that both primes are very large).

2.3.2 ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) was recently standardized as an asymmetric signing algorithm for DNSSEC in RFC 6605 [1]. ECDSA is not based on the prime

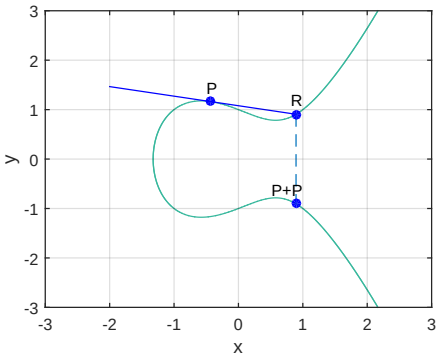


Figure 2.5: Example of adding point P to point P.

factorization problem (such as RSA), but rather on the elliptic curve discrete logarithm problem and belongs to the elliptic curve cryptography (ECC) family. In DNSSEC, calculations are performed on a set of curves with fixed parameters, the NIST curves. There are two curves used, P-256 and P-384, where the first curve provides shorter keys but the second one is cryptographically stronger. The NIST curves define a set of domain parameters which are required to compute with the elliptic curves. A curve E is defined by the equation:

$$E : y^2 = x^3 - 3x + b \tag{2.1}$$

The curve is defined over an algebraic group that consists of *integers modulo p*, where p is prime and its value varies per NIST curve. All points that satisfy Equation 2.1 belong to the elliptic curve. In addition, the point at infinity $\mathcal{O}$ is included as well. The coefficient b is pseudo-randomly defined and its value depends on the NIST curve used. A point G, consisting of an x and an y value, on curve E is used to generate the public keys. The order of G is denoted as n; in other words $nG = 0$.

In addition to the domain parameters, a scalar multiplication function needs to be defined in order to compute with elliptic curves. The group *self addition* for an elliptic curve is shown in Figure 2.5. When adding a point P to itself, the tangent line at P is used to find point R where the curve and the tangent line intersect. $P + P$ is defined as the reflection of R on the X axis. Group *multiplication*, i.e. kP, is defined as adding P to itself for k times. For example, $3P$ can be calculated by first adding P to itself $(P + P)$, as shown in Figure 2.5, and subsequently adding P to $P + P$ resulting in $P + P + P = 3P$.

The private key is chosen as an integer d mod n, such that d is in the range $[1, n - 1]$ ($d = 1$ is not allowed [21]). Using the domain parameters and the multiplication function defined above, the public key is computed as $Q = dG$. ECDSA's trapdoor mechanism is that given Q and G, it is extremely difficult to identify d: the elliptic curve discrete logarithm problem.

Signing and validation In ECDSA a signature of message m is created using curve E, domain parameters (G, n) and the private key d with the following steps [4]:

1. Select a random integer k, such that $1 \leq k \leq n - 1$.
2. $kG = (x_1, y_1)$ is computed, where x_1 is converted to the integer $\bar{x}_1$.
3. $r = \bar{x}_1 \mod n$ is computed. If $r = 0$, the algorithm starts over from step 1.

4. $k^{-1} \mod n$ is computed.
5. SHA(m) is computed and the resulting bit string is converted to the integer e. Depending on the NIST curve, the hash function is SHA-256 or SHA-384.
6. $s = k^{-1}(e + dr) \mod n$ is computed. If $s = 0$, the algorithm start over from step 1.
7. The resulting signature for message m is (r, s), where r and s are concatenated [1].

Both r and s are of approximately equal length as the prime p used to define the elliptic curve. This means that for P-256 and P-384 the signature size is $2 \times 256 = 512$ bits and 768 bits respectively.

The validation of an ECDSA signature requires the same curve E, same domain parameters (G, n) and the public key Q. With these parameters, the signature can be validated as follows:

1. The validating party should first check whether r and s are integers in the interval $[1, n-1]$ and whether Q is a point on curve E.
2. SHA(m) is computed and the resulting bit string is converted to the integer e.
3. $w = s^{-1} \mod n$ is computed.
4. $u_1 = ew \mod n$ and $u2 = rw \mod n$ are computed.
5. $X = u_1 G + u_2 Q$ is computed.
6. If $X = \mathcal{O}$, then the signature should be rejected. Otherwise, the x coordinate of X is converted to the integer $\bar{x}_1$ and $v = \bar{x}_1 \mod n$ is computed.
7. The signature is considered valid if $v = r$.

The dominant computation in ECDSA signature generation and verification is scalar multiplication (i.e. multiplying a curve point with an integer) [22]. The aforementioned steps show that during signature generation only one scalar multiplication is performed, while during the validation two scalar multiplications are performed. This indicates that signature validation in ECDSA is slower than signature generation.

2.3.3 EdDSA

The Edwards-curve Digital Signature Algorithm (EdDSA) is a relative young digital signature scheme that shares similarities with ECDSA [2]. Where ECDSA is based on the NIST curves P-256 and P-384, EdDSA is based on twisted Edwards curves, which are defined by the following equation:

$$-x^2 + y^2 = 1 + dx^2y^2$$

where also the addition and multiplication group functions are different from ECDSA. A particular twisted Edwards curve, Ed25519, was specifically selected for cryptographic uses, because it has several attractive features, including fast signature verification, small keys, small signatures and a high security level [2]. The curve's name is derived from the used prime, namely $2^{255} - 19$. Bernstein et al. show in [2] that their implementation of EdDSA outperforms other existing ECC signature algorithms.

2.4 Comparison

RSA, ECDSA and EdDSA each have their advantages and disadvantages. In this sections we discuss the relevant differences considering their usage in DNSSEC and compare their properties.

Table 2.1: Equivalent key strength (in bit length) [5].

Symmetric (in bits)	ECC (in bits)	RSA (in bits)
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15,360

Table 2.2: Key size, signature size and verification speed of RSA, ECDSA and EdDSA.

Signature algorithm	Key size (in bits)	Signature size (in bits)	Verification time (in cycles) [23]
RSA-1024	1024	1024	51,840
RSA-2048 [3]	2048	2048	105,408
ECDSA P-256 [1]	512	512	1,152,412
ECDSA P-384 [1]	768	768	4,345,540
Ed25519 [2]	256	512	271,372

In our comparison we include RSA-1024, RSA-2048, ECDSA P-256, ECDSA P-384 and Ed25519. RSA-2048 is the recommended version of RSA by NIST, where RSA-1024 is currently still in widespread use in DNSSEC. ECDSA P-256 and P-384 are both standardized in DNSSEC and are therefore included. We include Ed25519 as EdDSA implementation in the comparison because it was specifically designed with performance in mind.

Cryptographic strength Table 2.1 shows the equivalent key strengths of symmetric (e.g. AES), ECC and RSA algorithms. P-256 and Ed25519 (belonging to ECC) both have a group size of 256 bits, P-384 has a group size of 384 bits and RSA-2048 has a key size of 2048 bits. The table shows that both ECDSA schemes and EdDSA provide a better cryptographic strength than RSA. In addition, cryptographic attacks can be expected to become stronger in the future, so increasing the key size for future use should be a consideration. The table shows that elliptic curve key sizes increase linearly with cryptographic strength, whereas the RSA key size increase exponentially. Therefore, ECDSA and EdDSA are also more future-proof than RSA.

Key size The key size of the four schemes are displayed in Table 2.2. RSA-2048 keys are two times larger than RSA-1204, four times larger than P-256 keys, 5.33 times larger than P-384 keys, and eight times larger than Ed25519. The Ed25519 key is represented as its x-coordinate and a single bit providing the sign (positive of negative) of the corresponding y-coordinate. In contrast, the ECDSA keys consist of the full x- and y-coordinate, making the keys significantly larger than Ed25519.

Signature size Table 2.2 also shows the signature size for the signature algorithms. The signature size for RSA-2048 is twice the size of RSA-1024, four times larger than Ed25519 and P-256, and 2.67 times larger than P-384. Note that the signature size for RSA algorithms is equal to the cryptographic group size, where for the ECC schemes the signature size is twice the group size.

Validation speed In DNSSEC, signing zones occurs infrequently when compared to verification of a signature. Namely, an authoritative name server only signs an RR set when it is signed for the first time, when a signature expires or during a key rollover, whereas verification happens when a validating resolver receives an RRset. Therefore, we are particularly interested in the computation time of signature validation rather than signature generation.

In RSA, the public exponent e is often chosen as a small integer (1, 3 or 65537), where the private exponent is in order of 2048 bits in size. The public exponents are all prime, and have a small Hamming weight (i.e. the number of 1 bits) which significantly speeds up the exponentiation process. Therefore, RSA's signature verification is much less time consuming than signature generation. ECDSA and EdDSA do not have this favourable property, because the cryptography is not based on modular exponentiation.

According to RFC 6605, RSA supposedly signs messages *"20 times slower than ECDSA"*, but verifies signatures *"5 times faster in some implementations"* [1]. The ECRYPT Benchmarking of Asymmetric Systems (eBATS) project provides a set of measurements of public-key systems, including the verification time of RSA, ECDSA and Ed25519[2]. The verification time is expressed in the number of CPU cycles, and is measured on different computer systems. Table 2.2 shows the median of verification times for the different algorithms on the `hydra2` system. The `hydra2` system is one of the test systems used in the project and is the only one guaranteed to perform the validations on one thread [2]. RSA-1024 is capable of performing a single validation in the shortest amount of time and is roughly twice as fast as RSA-2048. P-256 and P-384 are a factor 22.2 and 83.8 slower than RSA-1024 respectively. EdDSA performs relatively better than ECDSA, but still is 5.2 times slower than RSA-1024.

The computation cost of signature verification is particularly important for validating DNS resolvers. These systems perform the verification action often and currently the whole DNS infrastructure is constructed with the computation cost of RSA in mind (or even without DNSSEC in mind). It is currently not known whether the existing DNS resolvers are able to cope with the increased computation of ECC when it is deployed on a wide scale.

Besides the aforementioned differences between RSA, ECDSA and EdDSA, there are many other differences. While we do not consider these difference important in our research in DNSSEC, we present them anyway and indicate why they are outside the scope of our research.

Signature generation ECDSA and EdDSA are capable of generating signatures much faster than RSA. However, this action is performed so sparsely compared to verification, that this is currently not considered to be a bottleneck in DNSSEC.

Key generation time Similarly to signature generation, keys are generated very sparsely to the verification of signatures. Again, ECDSA and EdDSA perform this action much faster than RSA.

Side-channel attacks In a side-channel attack, an attacker obtains information of the secret key by observing the physical implementation of an algorithm. For example, an implementation of a cryptosystem could use a different code path for each '1' bit in the private key, which would take just a little bit more time than the path for a '0' bit. An attacker could measure the run time of the system and extract the number of '1' bits in the private key. Previous research has shown that RSA [24] and ECC [25] without any countermeasures are vulnerable to these attacks. These attacks are in particular dangerous

[2] http://bench.cr.yp.to/ebats.html

for smart card applications [24]. In case of DNSSEC, all signing is performed on the authoritative name server and is in general not observable by an attacker. Hence, resistance against these attacks is not taken into consideration.

Random secret session key ECDSA generates a random k whenever a signature is created. The private key can be obtained when two signatures are generated with the same value of k, which occurs when k is static [26] or when a collision is generated [27]. Since key signatures are generated so sparsely, k collisions are unlikely to occur, assuming that the random number generation is implemented correctly [21].

2.5 Consequences

The comparison between the three cryptosystems shows that RSA is outperformed on almost all relevant aspects by ECDSA and EdDSA, except for the speed of verification. If the verification speed is tolerable, ECDSA could potentially be deployed in DNSSEC. This would have an effect on some current issues in DNSSEC, of which the two most important ones are described below: DDoS potential and fragmentation issues.

2.5.1 Distributed Denial of Service attacks

In a Denial of Service (DoS) attack, a victim is overwhelmed with traffic that floods its network connection. In order to make the mitigation of such an attack difficult, a Distributed Denial of Service (DDoS) attack is performed using an army of host nodes for generating network traffic. Depending on the victim, a substantial amount of network traffic needs to be generated to prevent the victim from serving its legitimate users. A DDoS attack can be used to shutdown a service that is reachable via the Internet, such as Web servers, DNS servers or a simple desktop computer. Recent attacks show that (the relatively easy to perform) DDoS attacks can have severe effects [28, 29, 30]. Two types of attacks can be distinguished:

i. The server's resources (such as sockets, CPU, HTTP connections, etc.) are exhausted; these attacks are generally performed by exploiting weaknesses on the application layer and are stealthier.

ii. The connectivity of a legitimate user is disrupted by targeting the network the user is connected to. Generally, these attacks are performed at the network or transport layer and are volumetric.

A particular devastating family of the second type attacks are *amplification* attacks. In an amplification attack, the bandwidth generated by the attacker is increased in size before it impacts the victim. An attacker does not directly send traffic to the victim, but rather sends the traffic to a large number of systems that reflect the traffic to the victim. These intermediate, reflective systems are referred to as *amplifiers*. An amplification attack allows an attacker who has limited access to bandwidth to still perform a powerful DDoS attack, or a powerful attacker to perform an extremely powerful attack. The ratio between the bandwidth generated by the amplifiers and the bandwidth generated by the attacker is defined as the *Amplification Attack Factor* (AAF):

$$AAF = \frac{\text{Amplified Attack Traffic}}{\text{Original Attack Traffic}}$$

A high AAF indicates a more devastating DDoS potential. Depending on the type of exploit used, an amplification attack can have an AAF of over several hundreds [31].

Name servers can be used in a DNS amplification attack as amplifier. If the name server hosts a DNSSEC signed zone, the amplification potential is even more severe. Consider a set of signed zones and an attacker that sends spoofed DNS requests to each of their corresponding name servers. Each NS will respond to the request with an answer and a set of large signatures and public keys. All response messages are sent to the spoofed IP address, i.e. the victim of the DDoS attack. The DNS requests are relatively small compared to the DNSSEC responses, so the AAF can be very large. Previous research by Van Rijswijk et al. [32] observed amplification factors of over 170.

While many countermeasures against DDoS attacks have been proposed, these attacks are still being performed with success. The AAF of an attack is crucial for the attack's success; if this value would be reduced, amplification attacks would be far less disruptive. A possible solution for DNSSEC-based amplification attacks would be to reduce the Amplified Attack Traffic, which would be achieved if ECDSA (or EdDSA) is deployed instead of RSA. Namely, the signature and key size of ECDSA is much smaller than RSA, and thus DNSSEC responses with ECDSA (or EdDSA) signatures are smaller in size than responses with RSA.

2.5.2 Fragmentation issues

A significant issue with DNSSEC is that firewall configurations may interfere with the workings of DNS. Firewalls may be configured to not forward fragmented UDP messages, which causes DNS resolvers behind these firewalls to not receive a DNS response. Van den Broek et al. [33] showed that around 36% of all DNS responses exceed the Maximum Transmission Unit (MTU) size (mostly the Ethernet MTU of 1500 bytes) and are fragmented at the authoritative name servers. In addition, up to 10% of the hosts are unable to handle fragmented packets. Since ECC reduces the size of signatures, keys and therefore DNSSEC responses, the deployment of ECC may reduce the experienced number of unresolved DNS queries due to firewall problems.

2.5.3 Combined Signing Key

The two issues described above are caused by large DNS messages. Another method of reducing the DNS message size is to reduce the number of `DNSKEY` records that are attached to DNS responses. This can be achieved by discarding the KSK and ZSK system and use a 'Combined Signing Key' (CSK). We recall the original reasons for adapting the KSK and ZSK:

i. The signatures in DNS responses should not be too large (i.e. the RSA key size should be small).

ii. The signatures should be sufficiently secure (i.e. the RSA key size should be large or a key rollover should be performed regularly).

iii. There should be as little parent-child interaction, because this takes place outside the DNS protocol (i.e. a key rollover should be performed irregularly).

The only way to fulfill these requirements is to adapt a large KSK that provides security and not a lot of parent-child interaction, where the small ZSK provides small signatures. However, Section 2.4 shows that ECDSA and EdDSA provide a more secure system than RSA (fulfilling requirement ii. and iii,), with a smaller key and signature size (fulfilling requirement i.). This indicates that a transition towards a CSK would not only simplify DNSSEC (which may be a reason for its small deployment percentage), but also would reduce the amplification factor of DNSSEC and reduce fragmentation issues.

Chapter 3

Problem statement & Goals

ECDSA and the cryptographically related algorithm EdDSA have significant advantages over RSA with respect to DNSSEC. The most notable advantages are the smaller key sizes, smaller signature sizes and better cryptographic security. One of the main arguments against deploying ECC globally as replacement of RSA is that it is currently unknown if validating DNS resolvers are able to cope with the higher computation effort caused by ECC. In addition, even if a transition towards ECC is currently possible, it remains unknown whether the current DNS infrastructure is able to cope with the computation load in future scenarios where much more domains are signed.

In order to solve the stated problem, the main question that is answered is as follows:

i. What is the effect of deploying ECC as a replacement of RSA in DNSSEC on the computation load of validating DNS resolvers?

During the process of answering the main question, the following sub questions also need to be answered:

i. What is the typical computation load of a DNS resolver?
ii. What is the typical behaviour of clients, resolvers and name servers?
iii. How can we model this behaviour and what are the relevant parameters?
iv. What scenarios can we evaluate to measure the impact of deploying ECC?

The first sub question requires the evaluation of a DNS resolver in the current state of DNS(SEC). The second sub question elaborates on that by also observing the behaviour of not only the resolver, but also the other relevant systems involved in DNSSEC: the clients and the authoritative name servers. By answering the third sub question, a translation from reality towards a DNS model is made by selecting the relevant parameters and translating them towards a model. The last sub question focuses on how the model eventually is used to evaluate the impact of deploying ECC as signature scheme.

3.1 Research goals

Our research is intended to provide a strong argument for or against the deployment of ECC as the replacement of RSA. As described in Section 2.4, RSA and ECC both have positive- as well as negative properties for cryptographic usage. One of the 'weak' properties of ECC is its presumed high validation cost in terms of computation power. If our research shows that the

validation cost of ECC poses no problem for the current DNS resolver infrastructure, choosing ECC over RSA would be advisable. We expect that our finding will pave the way for a worldwide deployment of ECC.

We create a model of the DNS infrastructure in order to answer our research questions. This model can potentially be further used and extended by future research that take upon the model-based approach. Currently, most research on DNS performance is measurement based, as opposed to model based. While measurements are valuable source material, they are often difficult and time consuming to obtain, so a model based approach could be valuable for future research.

Chapter 4

Related work

Since its development, DNS (and DNSSEC) has been popular for measurement based research. We discuss the most relevant research projects that are closely related to our own research for the various aspects of the project.

Computation load of resolvers in DNSSEC Several measurement-based research projects have focused on the performance of DNSSEC deployment. Migault et al. [34] performed several performance tests on different implementations of DNS and DNSSEC (BIND, NSD and Unbound) in terms of CPU cost and server response time. The current state of DNSSEC deployment (in percentages) is not taken into account; only the relative performance difference between DNS and DNSSEC is considered. Wijngaards et al. [35] performed a simulation where the performance of validating DNS resolvers was measured (the CPU load and the network traffic) for various percentages of DNSSEC deployment. Their research included the use RSA-SHA1 as signing scheme but did not include ECC. In contrast to our research, neither of these projects included ECC signature schemes.

The amplification factor of DNSSEC Anagnostopoulos et al. showed in [36] that an amplification factor of 44 could be achieved using DNSSEC DDoS amplification attacks. Further research by Van Rijswijk-Deij [32] gives a more in-depth view into DNSSEC and shows that an amplification factor of over 170 can be achieved. Both papers give a set of possible countermeasures, which include (among others) preventing IP spoofing using ingress filtering, reducing the response size, using EDNS0 cookies, limiting the response rate, disabling open DNS resolvers and detecting DNS amplification attacks. Rossow [37] proposes several countermeasures against UDP based amplification attacks, which are similar to the prevention methods by Anagnostopoulos and Van Rijwijk-Deij. Protocol hardening is proposed as an additional countermeasure, for example by introducing session handling to UDP. EDNS0 and UDP session handling provide stateful communication on the application layer and transport layer respectively. Most previous research focuses on eliminating the DDoS attack itself, but few mention that the reduction of the amplification factor is also a mitigation strategy. Rossow proposes a solution where the response size is of similar size as the request, which decreases the amplification factor heavily. Its downside is that the protocol efficiency drops and systems face higher loads in benign use. Van Rijswijk-Deij in turn proposes the reduction of the amplification factor by using cryptographic signature schemes with more favourable key and signature sizes.

Fragmentation related issues in DNSSEC Previous research on fragmentation [33] showed the problem caused by DNS fragmentation; 10.5% of all DNS resolvers encounter fragmentation problems. A possible proposed solution is limiting the response size, something that is provided by ECC signatures. Herzberg et al. [38] show that cache poisoning attacks based on fragmentation is possible. Ironically, DNSSEC is the main cause of fragmentation and this problem may be decreased by reducing the response size of DNSSEC responses.

Deploying a CSK The currently widely deployed RSA signature scheme uses a ZSK and KSK system to provide security, as well as the possibility to roll keys regularly without the undesirable parent-child interaction. Yang et. al [18] describe the design considerations that went into the separation of the two keys. Since ECC is cryptographically stronger than RSA while using smaller keys and signatures, we could potentially switch to a single-key signature scheme, using a *Combined Signing Key* (CSK). While there is some interest in supporting a CSK scheme [39, 40], there currently has been no actual research performed on a potential widespread deployment of such a scheme.

ECC deployment in DNSSEC ECDSA was standardized in DNSSEC in 2012 [1] and currently NIST recommends ECDSA as the preferred signing scheme from 2015 [41]. Furthermore, there are research efforts that explore the deployment of additional cryptographic schemes such as ECDSA in DNSSEC. Herzberg et al. [42, 43, 44] recognize the need for ECDSA and propose a negotiation protocol that enables DNS nodes to negotiate the used protocol, such that the DNSSEC overhead is minimized. While they do mention the issues with DNSSEC (the overhead) they do not address the performance considerations of deploying ECDSA.

Modeling of DNSSEC Jung et al. [45] performed a detailed analysis of DNS traces and measured the distribution of TTL values and domain name popularity, because (according to the authors) these determine the cache hit rate. Furthermore they performed a trace-driven simulation to study the impact of a shared cache on the hit rate. The authors used three DNS traces, which consisted of the queried domain names, the associated answers and the associated TTL values, for the simulation where the clients where grouped into groups of varying size sharing the same cache. The authors in [46] re-examine previous research (including [45]) and give in-depth measurements of DNS traffic, with a focus on malware detection. The results, which include a description of high level DNS characteristics, is interesting for our research since we try to model DNS traffic behavior. Koç et al. [47] attempted to capture the entire DNS in a model which according to the authors "*is intended to be used for analyzing 'what-if' scenarios*", exactly the purpose of our research. The research does not focus on DNSSEC in particular but is mostly a general model. Furthermore, Wessels et al. [48] performed laboratory simulations using a mini-Internet setup, where one client accesses one caching DNS resolver which in turn accesses three authoritative name servers (representing the root, TLD and SLD layers). Lastly, Kolkman [49] performed a research project similar to us. Before the deployment of DNSSEC (in 2005) they measured the effect of deploying DNSSEC on several metrics, e.g. CPU load. In contrast to our approach, he focused on the authoritative name servers rather than validating resolvers. He concluded that the name servers could easily cope with the increased CPU load, bandwidth and memory usage. Krishnan and Monrose [50] performed an empirical study on the performance effects of DNS 'prefetching' (performing DNS queries in advance, which is supposed to decrease the response time when browsing) and included a trace driven simulation to see the impact of prefetching in combination with DNSSEC. The used experimental setup included a validating resolver that played back a prerecorded trace and an authoritative name server that responded to the resolver's request with a correct answer. DNSSEC, and especially DNSSEC

combined with prefetching, resulted in a significantly higher response time and a lower throughput.

While these papers all use a (slightly) different approach in their research, there is not an existing DNS model from the resolver's perspective that is usable for us, because they either focus on DNS instead of DNSSEC or created the model from the name server's perspective.

Chapter 5

Methodology

As the main research question implies, the computational load of a validating resolver should be evaluated. Firstly, the current situation where the majority of the domains is signed with RSA could be compared with scenarios where ECC is deployed. In addition the long-term implications of deploying ECC should also be evaluated. The computational load of a resolver is determined by (1.) the computational load for a single signature validation and (2.) the number of signature validations that a resolver needs to perform. Previous work[1] has already extensively evaluated the performance for a single validation. In order to validate these results, a similar performance test has been performed as part of our research, which can be found in Section 8.1. The issue of determining the number of signature validations is more complex, because this depends on (among other) the live behaviour of the DNS, the configuration of the DNS resolver and the decisions made by domain owners.

5.1 Modelling versus simulation

We identified the following two approaches that allow us to evaluate the number of signature validations of a validating DNS resolver:

1. **A simulation approach** where in a controlled environment a simulated version of DNS is created. The actors in the environment should behave as realistic as possible in order to obtain valid results. The actors interact with each other using actual DNS messages. By changing the behaviour of the actors, different scenarios can be evaluated. Three possible actors could be a client, a resolver and an authoritative name server. This approach is similar to the approach used by Wijngaards et al. [35], who measured the effect of deploying DNSSEC.
2. **A statistical model-based approach** where a predictive model of DNS is created. Given a set of inputs, the model computes the expected number of validations. By changing parameters in the model, different scenarios can be evaluated. To the best of our knowledge, such an approach has not yet been used before in DNSSEC research.

The advantages (+) and disadvantages (-) of a simulation approach are stated in Tab. 5.1 and the (dis)advantages of a modelling approach are stated in Tab. 5.2. Since we are particularly interested in future scenarios (e.g. RSA is completely replaced by ECC or the DNSSEC deployment approaches 100%), modelling DNSSEC is the preferred approach for the research.

[1] http://bench.cr.yp.to/ebats.html

Table 5.1: The advantages and disadvantages of a simulation approach.

+/-	**(Dis)advantage**
+	A default simulation setup is relatively easily deployed. Client behaviour could potentially be simulated by replaying a pre-measured trace (e.g. using `tcpdump` and `tcpreplay`), where the authoritative name servers simply reply with a valid or invalid response. The CPU load of the resolver can be measured in real time.
-	Deviating from the current, default behaviour is not trivial. Many assumptions need to be made which may result in unrepresentative situations. For instance, when increasing the query traffic from clients to the resolver, what happens to the distribution of domain names? For answering the research question, it is crucial that non-standard situations can be evaluated, so a simulation approach may not be sufficient.
-	In order to obtain meaningful results from a simulation, it (presumably) needs to run for a relatively long time compared to a modelling approach. In case of many scenarios, it may be infeasible to run all these simulations within a reasonable time span.

Table 5.2: The advantages and disadvantages of a modelling approach.

+/-	**(Dis)advantage**
+	Once the model and its parameters are established, it is a powerful tool for predicting scenarios. For example, an arbitrary resolver administrator can estimate the number of signature validations without the need of setting up an extensive simulation setup for himself.
-	Compared to a simulation approach it requires intensive measurements to determine the correct parameters and variables for the model.
-	The model should be carefully developed. If too many details of DNS are included (i.e. reality is modelled too closely), the model loses its predictive property since it is purely trained for its test data, rather than non-test data. If too little details are incorporated, the model may provide false results or may not reflect reality at all.

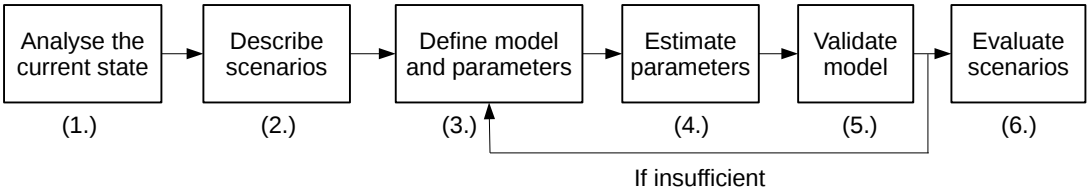


Figure 5.1: Research steps

5.2 Modelling approach

In order to create a model that could be used for prediction, intermediate steps were required that led to a complete model. Fig. 5.1 shows the research methodology steps that were being followed. A more detailed description is as follows:

1. **Analysis of current state** Before any attempt at modelling was made, an in-depth analysis in the current state of DNSSEC was performed. Based on measurements on three resolvers of SURFnet this gave a basic understanding of DNS. Since data was gathered at resolvers, all metrics are defined from the perspective of the resolver. For example, incoming queries refers to the queries that are sent from clients towards a resolver, where outgoing queries refers to the queries that the resolver sends to authoritative name servers. Examples of measured variables are incoming queries per second, incoming signatures per second, signature validations per second, distribution of domain name popularity, etc. These measurements also gave an indication of what parameters should be included and excluded from the model.
2. **Scenario description** Before developing a DNSSEC model, it is crucial to think of potential scenarios that need to be validated. Namely, variables that change in these scenarios should be incorporated in the model. These described scenarios reflect the worst case scenarios and realistic scenarios that are (and will be) encountered.
3. **Defining the scope of the model** In this step, the input, the output and the parameters of the model were determined. A major priority was defining the model in such a manner that reality can be approximated sufficiently, without creating a very specific model rendering it unusable for scenario evaluation. The parameters were chosen such that they are independent of each other (i.e. any of the parameters can be changed without affecting the other parameters). Additionally, the model includes a set of functions that describe how the input and the output of the model relate to each other.
4. **Parameter estimation** Based on a set of measurements, the parameters were estimated using regression techniques. This set of measurements is different from the measurements from the current state analysis (e.g. the new data set measured more metrics). This and the previous step were performed more than once to improve the model until it was sufficiently effective in predicting the output based on the input. After this step, the model was usable for scenario evaluation.
5. **Model validation** After the parameter estimation step, the quality of the model was evaluated. We assessed how well the model predicts the number of validations required by comparing the prediction to an actual measured number of validations. If the model does not represent the current state of DNSSEC sufficiently, it cannot be used for predictions. In case of an insufficient model, the model was redefined and its parameters were re-estimated.

This iteration step was performed several times during the project.

6. **Scenario evaluation** The scenarios described in step (2.) were evaluated using the developed model. Based on these scenarios, the research question could be answered.

Chapter 6

Analysis of the current state

In this chapter, the findings of a first, exploratory measurement session are described. The reason for performing this measurement was to get basic insight in the usual behaviour of DNS. This default setting allowed us to determine what parameters could potentially be included in - or excluded from - the eventual model.

6.1 Measurement setup

In order to get the necessary data, the network traffic of three validating DNS resolvers was captured. These resolvers are operated by SURFnet and are used by the Dutch research and education network. For the remainder of the report, we refer to these resolvers as AMS, TIL and UTR, named after the cities 'Amsterdam', 'Tilburg' and 'Utrecht' where the resolvers are deployed respectively.

Fig. 6.1 displays the key components in the used measurement setup. The regular DNS infrastructure consists of the standard DNS systems: clients, a single resolver (AMS, UTR or TIL) and the pool of authoritative name servers. All network traffic between clients and the resolver, as well as the traffic between the resolver and authoritative name servers, is tapped by sensors and collected by the multiplexer (MUX). The MUX retransmits the tapped data to another system, which performs the actual data extraction from the network traffic.

6.1.1 Data extraction

For the data extraction, two tools were used: the 'Extensible Ethernet MOnitor' (or Eemo) and Unbound. Both tools ran simultaneously and were instructed to regularly output several metrics of interest. Eemo[1] is an open-source network measurement capable of monitoring network traffic in real-time. As the name implies, the tool allows the development of additional plugins (in C) that extend the capabilities of the tool. Eemo support the processing of different network layers and protocols, including IP, UDP, TCP and DNS(SEC). For the exploratory measurement setup, a new plugin has been written, which can be found on Github[2] under the name `dnsdistribution`. Among the gathered statistics are the following metrics:

- Distribution of requested domain names

[1] www.github.com/SURFnet/eemo
[2] www.github.com/kdhageman/eemo

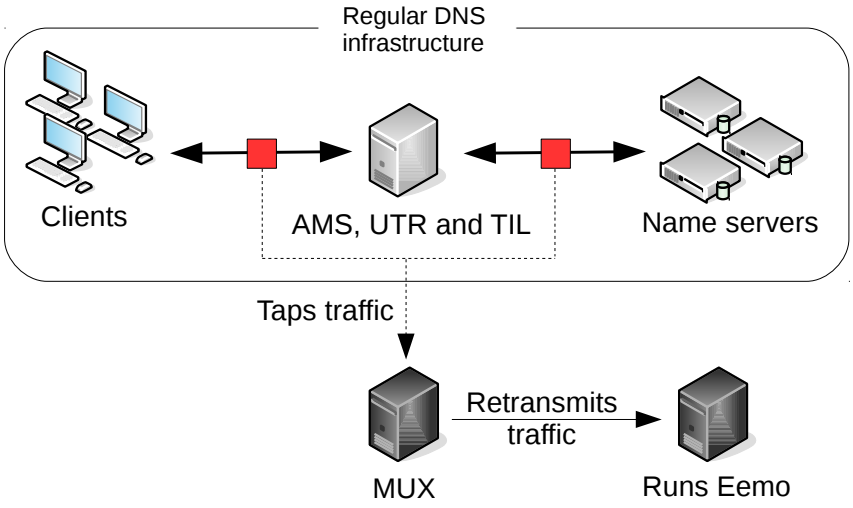


Figure 6.1: Exploratory measurement setup

- Distribution of unique TTL values
- Cache hit ratio
- Signatures per signed response
- Number of received signatures per second

Besides these statistics measured by Eemo, we were interested in others as well. In particular, the following metrics needed to be measured:

- The number of signatures validations (per second)
- The distribution of algorithms and key sizes for validations

Depending on the characteristics of an incoming signature and the implementation of the DNS software, the signature may or may not be validated. Simply measuring the occurrence of a signature in the DNS traffic does not guarantee an accurate measurement of the number of signature validations. Measuring the actual number of signature validations requires access to the internal workings of DNS resolver software. We chose to use Unbound[3] as the resolver software and slightly modified it to output our metrics of interest at regular intervals. The customized code can be found in Appendix A. Unbound is an open source software tool developed by NLnet Labs and is used by SURFnet on their resolvers. The main alternative to Unbound would potentially be BIND[4], which is the most widely deployed DNS server. Both Unbound and BIND are open source, which allows for altering their source code for measurement purposes. However, Unbound was chosen over BIND due to the experience of SURFnet with Unbound.

All statistics were measured over a period of seven days, where the data was written in ten minute intervals. This means that all the data presented in this chapter as 'average per second' is the average of a ten minute time window. The reason that the data was written every ten minutes rather than more often is that the storage of the computer system was limited. A measurement of

[3]https://www.unbound.net/
[4]https://www.isc.org/downloads/bind/

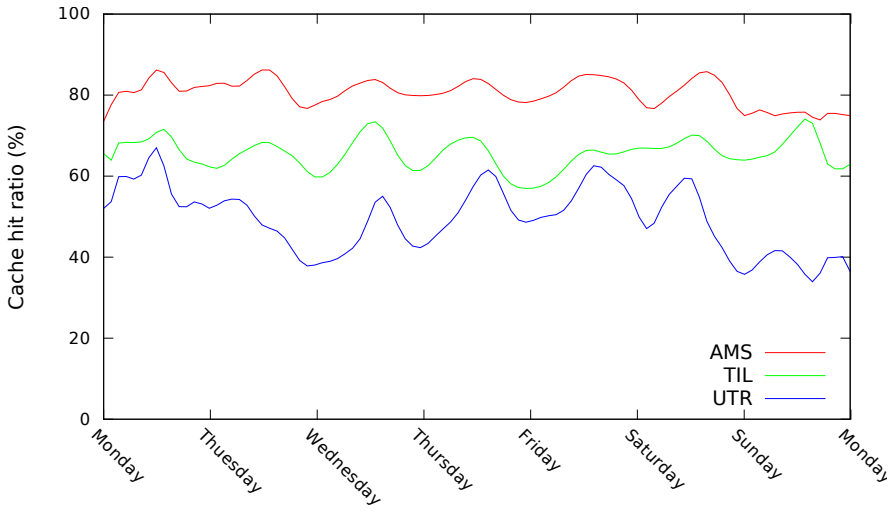


Figure 6.2: Cache hit ratio for three resolvers over time. The ticks on the x-axis represent 4 PM for the different days.

a relative long period (i.e. seven days) was required for an accurate TTL measurement. Namely, due to the caching mechanism of a resolver, a record with a TTL value of one week will at maximum exist once in a measurement of a single week. By measuring only a single day, the number of records with a high TTL value will be heavily misrepresented in the measurement. A measurement of much longer than one week would not be feasible due to the total duration of the project.

6.2 Results

From the resulting data, features were extracted and two major conclusions regarding the model could be made. These conclusions involve the cache hit ratio of a validating resolver and the current state of DNSSEC deployment.

6.2.1 Cache hit ratio

One of the key goals of the measurements was to get insight in the caching behaviour of a resolver. The cache hit ratio (CHR) indicates how many incoming queries (from clients) result in outgoing queries (towards authoritative name servers) and thus potentially in signature validations. We identified three elements that affect the cache hit ratio of a DNS resolver:

- Incoming queries per second
- Distribution of domain names of these queries (i.e. domain name popularity)
- Associated TTL value of the domain names

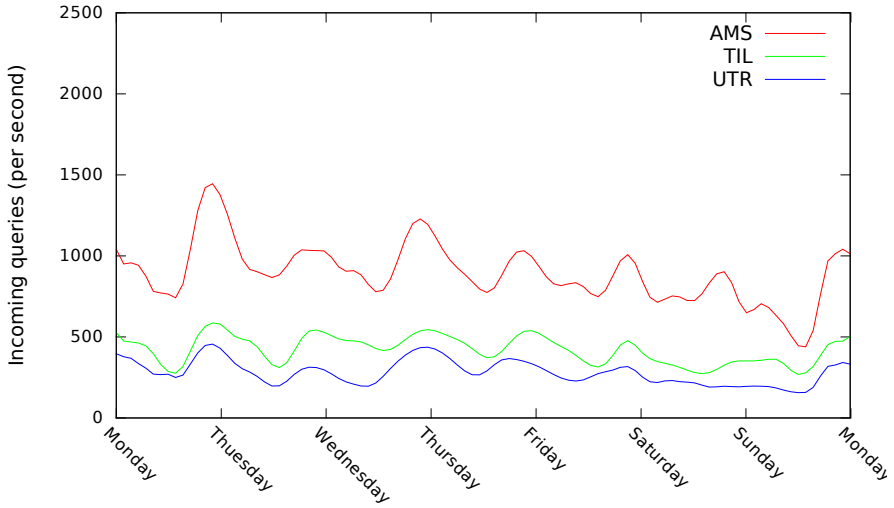
To give an indication of realistic cache hit ratio values, Fig. 6.2 displays the CHR of the three resolvers over time. Notably, the CHR of the different resolvers heavily differ on average, with AMS having the highest average of 80.49, TIL having an average of 65.52 and UTR having the lowest average of 49.29. Additionally the CHR of the UTR resolver seems to deviate much more

from its average than the other two resolvers. For AMS and UTR, in the weekend (in particular Sunday) a lower cache hit ratio can be observed. This may be caused by the lower amount of incoming query traffic, which implies that the inter-arrival time of queries for a particular domain name is longer. The likelihood that during the inter-arrival the TTL is expired is therefore increased and thus the CHR increases.

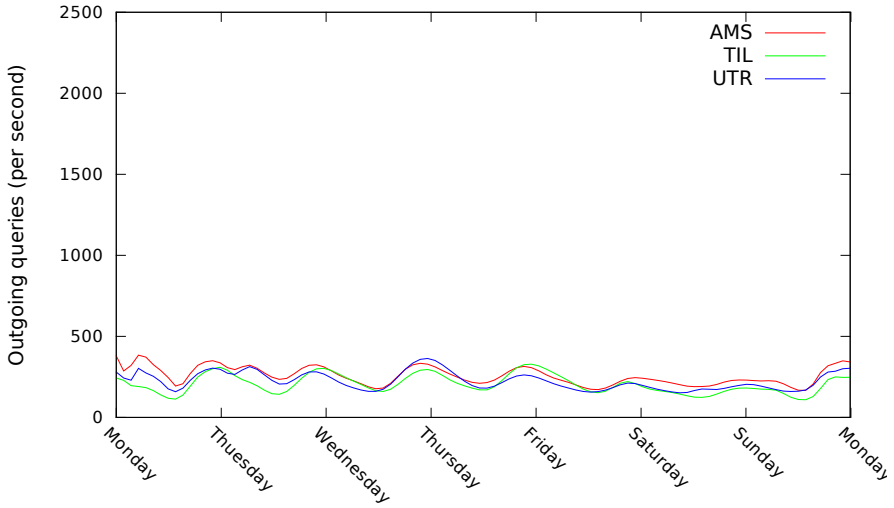
Incoming queries per second Fig. 6.3a shows the average incoming messages per second for the three resolvers. A day and night pattern can be observed, where the daily peak is at around 12:00h. In addition, on weekend days the query load is less than during week days. These results correspond with the user base of SURFnet's resolvers: employees and students of research- and educational institutes are more likely to use DNS based software during the work- and study hours. Fig. 6.3b shows the average *outgoing* query load for the same resolvers and the same time window. Interestingly, all three resolvers (who see very different incoming query loads), have a very similar amount of outgoing queries. This confirms the result from Fig. 6.2 that all three resolvers have a different CHR, with AMS having the highest and UTR having the lowest. Example 6.1 demonstrates how two resolvers with a different incoming query load may share a similar amount of outgoing queries.

Example 6.1. Consider two resolvers, A and B. Both resolvers receive only queries requesting the `A` record for a single domain, namely `www.example.com`. For the sake of the example, the `A` RR set for `www.example.com` has a TTL value of 240 seconds. Resolver A receives queries at a rate of one query per 30 seconds, while resolver B receives them at a rate of one query per 60 seconds.

The figure below shows the effect of the TTL and the caching behaviour of resolvers on the number of outgoing queries. A timestamp t_0, both resolvers have an empty cache. At t_1, the first query for `www.example.org` arrives, which is for both resolvers a cache miss. In case of a cache miss, the resolvers attempt to resolve the query by recursively request an answer from authoritative name servers. Then, accordingly to the received TTL value, both resolvers set t_{10} as the expiration time of the cached entry. All queries that are received between t_1 and t_{10} are immediately answered from the cache. As the figure shows, resolver A receives seven in the intermediary time period, while resolver B receives three queries in the same amount of time. The queries received at t_{10} cannot be answered from the cache (since its TTL expired) and thus the resolvers forward the query towards the authoritative name servers.

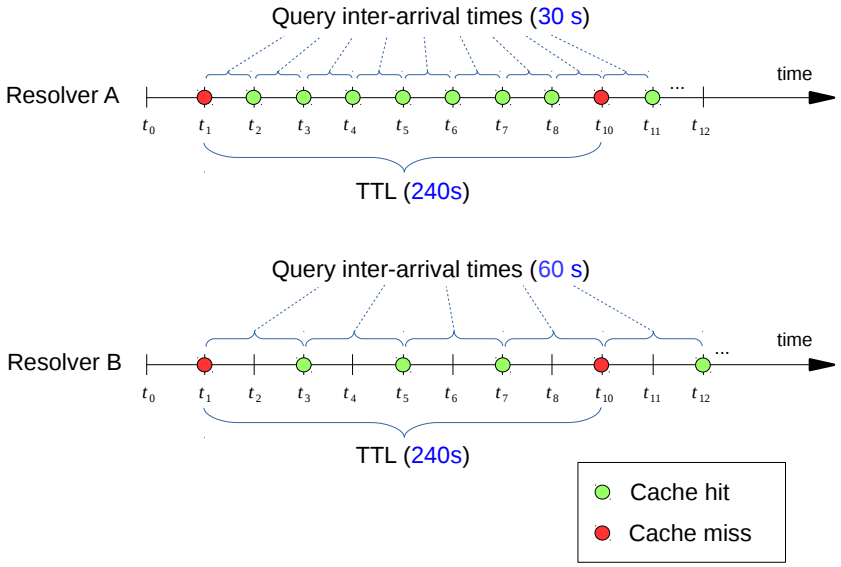


(a) Incoming queries



(b) Outgoing queries

Figure 6.3: Traffic of AMS, TIL and UTR over time.



If this pattern would repeat itself, it is clear that although both resolvers see different amounts of incoming queries, the amount of outgoing queries is the same. In this case, resolver A has a cache hit ratio of 87.5% and B's cache hit ratio is only 75%.

TTL distribution As the above mentioned example indicates, the TTL value of a resource record impacts the number of outgoing queries for that particular domain. For this reason, the TTL distribution of all unique resource records in DNS responses was measured (shown in Fig. 6.4). Note that the x-axis is logarithmic rather than linear. Again, the data in the plot was extracted from a seven-day measurement. The most common TTL values (ordered by frequency) are one day, 5 minutes, one hour, two days and ten minutes. The received TTL values of AMS are generally shorter than those of TIL and UTR. The default configuration of Unbound states that cache entries are dropped from the cache automatically after one day. Resource records with a TTL higher than one day therefore can be considered to have a TTL of one day.

Domain name popularity As our example showed before, the inter-arrival between two queries requesting the same RR set in relation to the TTL value determines the cache hit ratio. Maintaining a table of inter-arrival time for every domain name was considered infeasible. Therefore, rather than the inter-arrival times, the total number of requests for a particular domain name has been measured. Based on this number, an average inter-arrival time can be easily computed and is assumed to be the inter-arrival time between queries. The total number of requests for a domain name is for the rest of the report referred to as *domain name popularity*. Fig. 6.5 shows the domain name popularity obtained from the measurements at the three resolvers. Note that both the x- and the y-axis are logarithmic. The domain names are ranked based on their popularity, where the first ranked domain name is most popular and the last ranked domain name is least popular (i.e. is only requested once). In these three data sets, the highest number of domains was measured in the AMS data set, which contained 12.5 million domain names. The domain name popularity distribution seems almost linear (in a loglog plot) as shown by the black dashed line, which confirms earlier studies in domain name popularity. This type of distribution (the Zipf-distribution) can be used to describe the distribution of unique words in an arbitrary

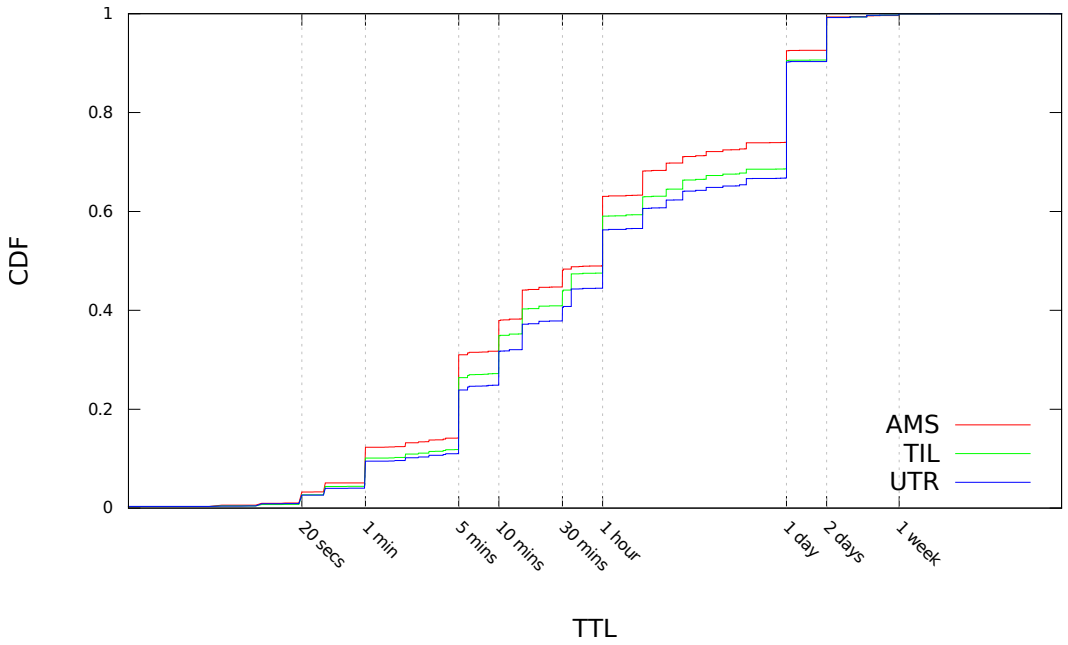


Figure 6.4: CDF of unique TTL values for three resolvers.

textbook or resources on the Internet in general [45]. In our case, the top 0.5% popular domains seem to follow a different popularity distribution than the remaining domains. Around this point, many domain names share the same popularity.

In theory, by combining the popularity of all requested domain names and their TTL values, the exact content and volume of the outgoing queries can be computed. Therefore, it could be used as the input of our model of DNSSEC. However, measuring all this data is not only infeasible, but possibly also very resolver specific. This indicates that the model (once parametrized) can only be used for a single resolver. Since we wish to offer a simple method for an arbitrary resolver administrator to predict its computation load, this is not an acceptable approach. It might be a better approach to use the outgoing query traffic volume as an input for the model.

6.3 State of DNSSEC usage

Although the DNSSEC standard has existed for over 10 years now and the root has been signed since 2010, the current usage of DNSSEC is still not widespread. In addition, most of the used signatures nowadays are RSA rather than ECDSA. In the exploratory measurement, the exact state of the DNSSEC usage was measured to quantify the possible growth of ECDSA usage. This growth is possible in two ways: deploying ECC instead of RSA or increasing the overall DNSSEC deployment.

Table 6.1: Current usage of cryptographic algorithms and their key sizes

Algorithm	Key size	Occurrence (%)
RSA	1024	89.1793
	2048	9.1468
	1280	0.8018
	1536	0.3908
	4096	0.2180
	1048	0.1407
ECDSA	512	0.0368
RSA	1032	0.0293
	944	0.0234
	512	0.0174
	2560	0.0051
	1152	0.0024
	936	0.0024
	768	0.0012
	1304	0.0010
	2304	0.0009
	3072	0.0005
	2024	0.0004
DSA	3240	0.0004
RSA	2096	0.0003
	928	0.0001
DSA	1704	0.0001
RSA	2088	0.0001
	1160	0.0001
	2016	0.0001
	1400	0.0001
ECDSA	768	0.0001
RSA	904	0.0000

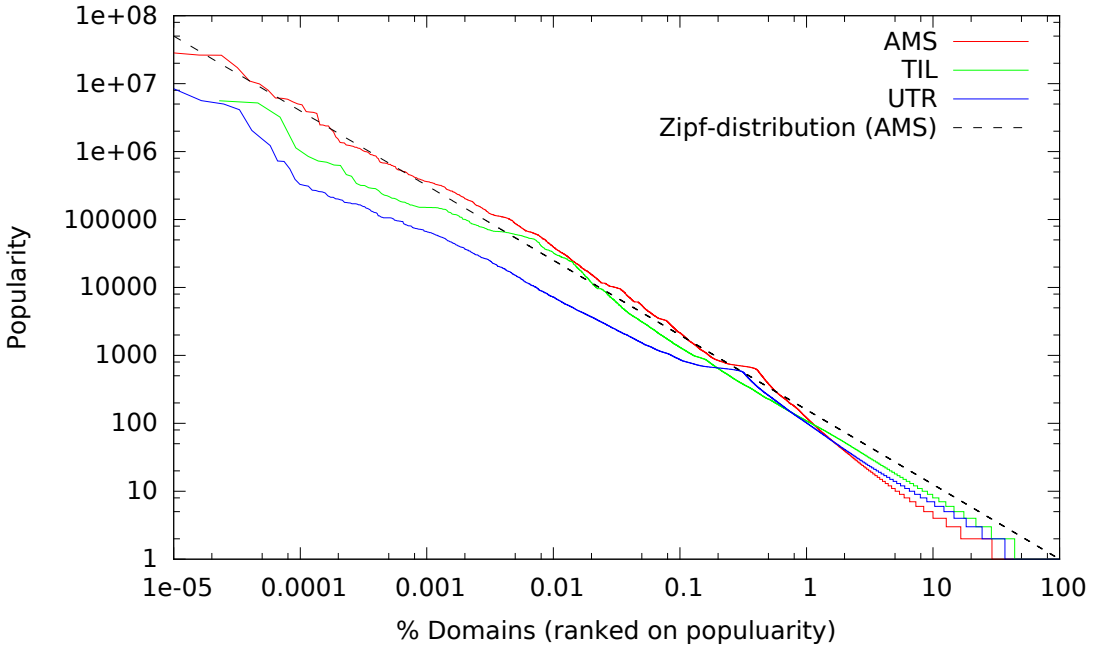


Figure 6.5: Query name popularity distribution (resembling the Zipf-distribution)

6.3.1 Cryptographic algorithm usage

The current algorithm usage was measured using Unbound rather than Eemo. Using Eemo, only the received signatures can be measured, but not which of these signatures is actually validated. Only the signature validation is of interest (rather than receiving the signature). In Unbound, the exact code block in which a validation takes place was extended with monitoring code to measure the used cryptographic algorithm and its key size.

Tab. 6.1 shows the distribution of algorithms and their key sizes. The table shows the results from measurements on a single resolver (AMS), but other resolvers show a similar distribution of algorithms. There is no distinction made between the different RSA algorithm numbers, which use different hash functions, since the computation effort of a hash function is negligible compared to a signature validation and therefore these different algorithms are equivalent to each other. Clearly, ECDSA is almost non-existent compared to RSA: only 0.037% of all signatures is signed using ECDSA. Between ECDSA P-256 (with a key size of 512 bits) and ECDSA P-384 (with a key size of 768 bits), P-384 was rarely used. In fact, only seven signature validations were performed by Unbound over the course of the measurement. An overwhelming majority of all signatures is generated using RSA with key size 1024, an algorithm that is considered to be unsafe and according to NIST should be replaced by keys of size 2048 in 2013 (which was extended to the end of this year) [51]. Besides 1024 and 2048, there are some odd RSA key sizes, such as 1152, 936 and 768. Since RSA is able to work with arbitrary key sizes, nothing prevents zone administrators from signing their records with keys of exotic sizes. Lastly, the table shows that the Digital Signature Algorithm (DSA), another alternative to RSA, is also rarely used.

From the table can be concluded that ECC is currently barely being used in DNSSEC as signature algorithm, which indicates that a major growth in the CPU load of validating resolvers is still possible.

6.3.2 DNSSEC deployment

Besides replacing RSA with ECC, an increase in the total deployment of DNSSEC will also contribute to an increased CPU load of a resolver. The DNSSEC deployment is the percentage of domains that is signed. Currently there is no entity that keeps track of the global DNSSEC deployment. By combining different sources, an estimation of the deployment can be made. The DNSSEC Deployment Report[5] maintains a table of all TLDs and their associated DNSSEC deployment status. Additionally, the deployment percentage of many signed TLDs is stated as well. Unfortunately, the deployment of several important TLDs is not stated, such as `no.` (Norway), `uk.` (United Kingdom) and `eu.` (Europe). Currently, 84% of all 1,054 TLDs are signed. DomainTools[6] states the zones size (i.e. the number of domains per zone) for all TLDs. Using Eq. 6.1 an estimation of the total DNSSEC deployment (D) can be made, where i refers to the TLD, $n(i)$ is the number of domains in TLD i, $d(i)$ is the DNSSEC deployment percentage for TLD i and t is the total number of domains worldwide.

$$D = \sum_{i \in TLDs} \frac{n(i) \times d(i)}{t} \tag{6.1}$$

Essentially, the equation computes the weighted average of the DNSSEC deployment per TLD, where the weight is defined by the number of zones in the TLD. If the DNSSEC deployment of TLDs such as `no.` is assumed to be 0%, a lower bound of the current DNSSEC deployment can be estimated: $D = 1.685\%$. Previous estimations of DNSSEC [52] provide a DNSSEC deployment percentage of 3%. In the theoretical scenario where every single domain in the world is signed, a validating resolver will encounter a larger CPU utilization than at the moment.

6.4 Conclusions

Based on the exploratory measurements, the following conclusion can be drawn:

- Although the cache hit ratio has a major impact on the number of outgoing queries and signature validation, we feel that including the relationship between incoming queries, cache ratio, domain name popularity and TTL values can be omitted from the DNSSEC model. Instead the number of outgoing queries can be used as the input of the model. This omits TTL values completely from the model, because once a query is sent from the resolver towards the name servers, the TTL of the returned resource record has no short-term effect on the number of signature validations.
- There is still a tremendous growth possible in the usage of ECC signatures in DNSSEC. Firstly, since the current usage of ECC is so sparse, replacing RSA with ECC signatures will already heavily increase the number of signature validations with ECC. Additionally, the number of signed domains has the potential to grow from the current deployment of 1.685-3% of all domains to 100%. Therefore, a future scenario should be validated.

[5] http://rick.eng.br/dnssecstat/
[6] http://research.domaintools.com/statistics/tld-counts/

Chapter 7

Model

In order to answer our main research question, a model based approach was chosen because it enables the scaling of our validation to future scenarios, without the need of an extensive simulation setup. For instance, given a model of DNSSEC it is possible to evaluate the scenario where every single domain in the DNS is signed. The developed model is statistical in nature, i.e. given a set of inputs, the output is computed using a set of functions and probability distributions. Since the computation load of a validating resolver is assumed to be dominated by its signature validations, the output of the model was chosen to be the number of performed validations per second. The output of the model is computed in several steps. The components and their mutual interaction from Fig. 7.1 show the steps from the input of the model towards the output. A detailed description about the model is as follows:

- The model assumes that a validating DNS resolver initiates DNS lookups on behalf of its clients. As the exploratory phase indicated, translating the behaviour of clients to the outgoing queries of a resolver is infeasible and is excluded from the model. The outgoing queries per second, denoted as Q, is therefore considered the input variable of the model.
- The authoritative name servers respond to the queries with referrals and authoritative answers. The total response traffic from the name servers to the resolver is denoted as R. A single outgoing query does not necessarily result in a response, because UDP (the transport layer protocol that DNS is generally used over) does not guarantee that packets between two hosts are successfully transmitted. Queries from the resolver may not arrive at the name servers, servers may not be available and responses may be lost over the network as well. In addition, in case of a time out the DNS software may initiate retries, which increases the number of outgoing queries while not necessarily increasing the number of responses. In conclusion, the relation between Q and R needs to be established using measurements and R is considered to be a mathematical function of Q (①).
- Only a fraction of R will contain signatures. The signed responses (i.e. responses containing signatures) are referred to as R_s. The remaining responses contain no signatures and are referred to as R_n. The fraction of responses having signatures is determined by the fraction of domains being signed and the popularity distribution of these domains. R_s can also be defined as a mathematical function, but in this case a function of R (②).
- By multiplying the number of signed responses (R_s) with the average signatures per signed responses, the total number of incoming signatures can be obtained. We assume that in a future scenario, the average signatures per signed response remains the same as it does now. Namely, the signatures in a signed response can be classified in two groups: (1.) the

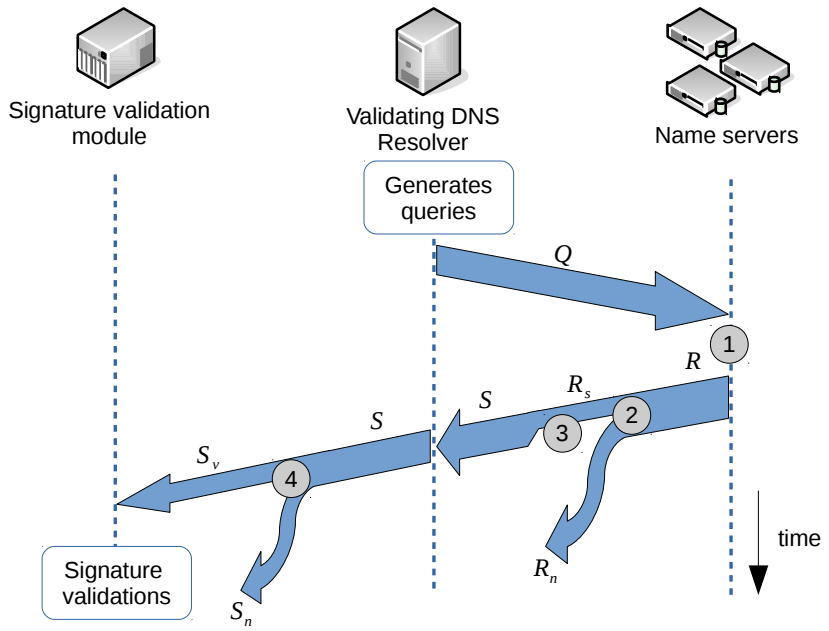


Figure 7.1: Components of the DNSSEC model and their mutual interaction.

authoritative data and (2.) non-authoritative data. In the first group the signatures for the requested RR set are given. The second group contains signatures for additional records, such as `NS` records. Both groups are unlikely to change if the number of signed domains increases or if the cryptographic algorithm changes. The average number of signatures per signed response needs to be determined using measurements, and in the model is assumed that S is a function of R_s (③).

- Not all incoming signatures will be validated by the resolver. Several reasons exist for not validating an incoming signature, including the following:

  1. The signature is part of a non-authoritative answer and is not validated by Unbound. Whether other DNS resolver software tools also will not validate these signatures is unknown.
  2. A chain of trust for a particular signature cannot be established.
  3. A signature is already cached. For example, a duplicate answer from two different authoritative name servers arrive at the resolver.

  Therefore, a fraction (between 0 and 1) of all incoming signatures (S) will be validated. The number of performed validations is denoted by S_v, where the number of non-validated signatures is denoted as S_n. Similarly to the other variables in the model, a mathematical function can be defined that maps S to S_v (④). From the above stated list, item (1.) and (3.) are unlikely to change in the near future. The second item however may change, because due to the increasing deployment of DNSSEC, the number of domains for a chain of trust that does not exists is getting smaller. Also, the domains that have signatures but

no valid chain of trust may mainly reside in a particular TLD. Resolvers that perform many domain name lookups for that particular TLD may therefore validate relatively a small percentage of incoming signatures compared to resolvers that do not query that TLD.

The model description above states that there exist four functions that are able to 'predict' a value based on a single input. For example, given the number of incoming signatures per second, there exists a function that predicts the number of signature validations. Due to the complexity and many uncertainties of DNSSEC (e.g. packet loss, delays, etc.) it is infeasible to accurately describe these function mathematically. Therefore, approximations of these functions have been devised. Besides one or more inputs, the functions require a set of parameters. These parameters have been estimated using regression, and with the right parameters, our model should be able to predict the number of signature validations.

7.1 Parameter estimation

Before the model could be used for predicting the CPU load for future scenarios, the parameters of four functions needed to be estimated through measurements. Before any measurements were performed, the shape of these functions were unknown. Therefore, the following approach was used to obtain the parametrized functions:

1. Measure the metrics of interest: Outgoing queries (Q), incoming responses (R), incoming signed responses (R_s), incoming signatures (S) and signature validations (S_v), as they were described in the previous chapter.
2. Plot the (presumably) correlated variables against each other in a two dimensional scatter plot.
3. Determine what type of function could estimate the trend in the observed data. This also introduces the parameters that need to be estimated.
4. Use regression techniques to fit the determined function as closely on the data as possible.

The result of this approach would be four functions that describe how two variables relate two each other. By chaining these functions together, the number of signature validations per second (i.e. the output) can be predicted based on the number of outgoing queries per second (i.e. the input).

7.1.1 Measurement setup

The measurement setup for the parameter estimation was very similar to the exploratory measurement setup from Section 6. Fig. 7.2 shows the involved systems. The regular systems in the DNS interact with each other: clients, a resolver and the authoritative name servers. A multiplexer (MUX) taps all traffic from the AMS, UTR or TIL resolver. Where the MUX was instrumented to retransmit all network traffic towards an analysis system running Eemo, the MUX in this case only retransmitted the incoming queries towards a system running Unbound. In other words, the MUX acts as client population for the analysis system and the requested queries are identical to the queries that AMS, UTR or TIL receive. The analysis machine running Unbound performed DNS lookups on behalf of the MUX and thus interacted with the authoritative name servers. Besides Unbound, the system also ran Eemo that measured four variables of interest: Q, R, R_s and S. In addition, Unbound measured the missing variable: S_v. In total the measurement contains 5040 data points, corresponding to seven days of measurement data. Each data point contains the average of a 120 second time window.

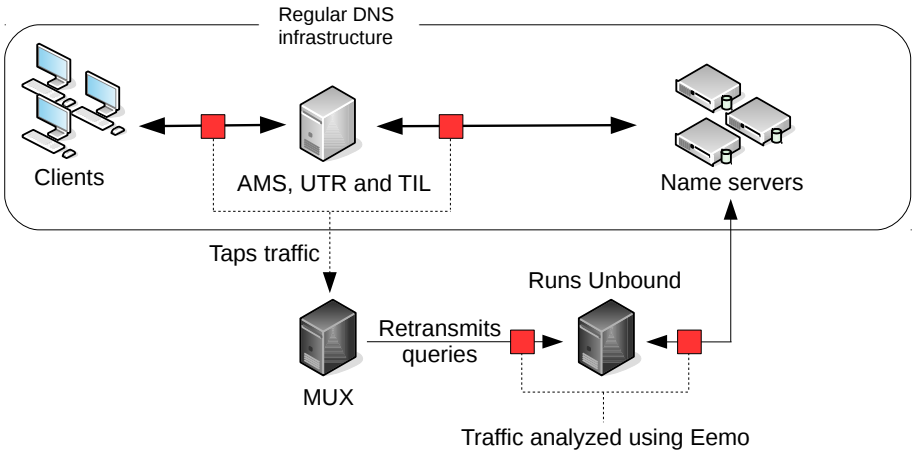


Figure 7.2: Parameter estimation measurement setup.

7.1.2 Correlation between the variables

Fig. 7.3 shows four scatter plots where the correlation between pairs of variables is shown. The measurements have been performed on three different resolvers, and the three colors in the plots indicate data points from the different resolvers. The plots are drawn on top of each other, so the data points of `UTR` partially obscure the data points of `AMS` and `TIL`. The left top plot shows the correlation between Q and R, and by quickly glancing over the figure, it is clear that a strong linear correlation exists between the number of outgoing queries and the number of responses to those queries. One would expect that all queries result in a single response, but in reality the unreliable UDP protocol does not guarantee delivery of packets and thus responses may not arrive at the resolver. A linear function in the form of $R = aQ$ would in theory describe the data the best, where the parameter a represents the average number of responses per query. However, using a linear function in the form of $R = aQ + b$ might be better, because a regression method will find a better fit on the data.

In the right top plot, the correlation between the number of responses and the number of signed responses is displayed. Again, a positive, linear relation seems to exist between the two variables, although the relation is not as strong as in the previous figure. In particular, the data of the `UTR` resolver has outliers. Still, a linear function in the form of $R_s = aR + b$ might be sufficient for data fitting. The parameter a represents the fraction of responses containing signed data.

The relation between the number of signed responses and the number of signatures is displayed in the left bottom plot. Similarly to the other figures, a positive linear trend can be observed. The `UTR` resolvers has a few outliers lying above the linear trend line, in particular when the input of the function is large. In the resulting linear function $S = aR_s + b$, the parameter a represents the average number of signatures per second.

The last figure (the right bottom plot) shows the correlation between the number of incoming signatures and the number of signature validations. The slope of the fitted linear function ($S_v = aS + B$) approximates the true fraction of signatures being validated.

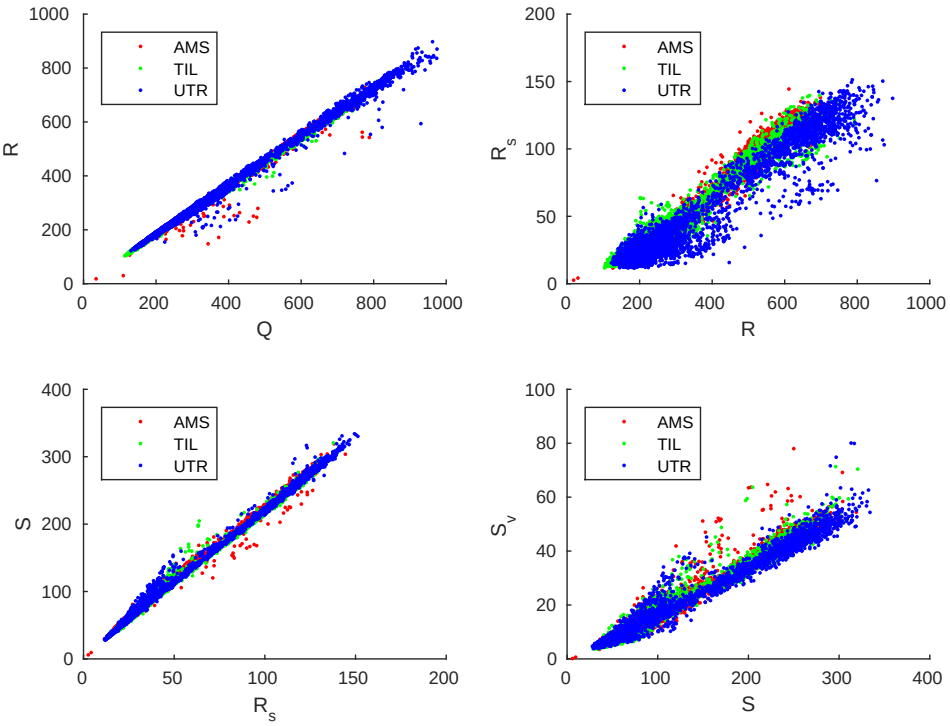


Figure 7.3: Correlations between the variables

All four functions appear to be a linear function. A linear function that intersects the data points (0,0) would describe reality the best, because for all those functions holds that if $x = 0$, $y = 0$ also must hold. For instance, if no signatures are received per second, exactly no signature validations take place. However, in order to get the best data fit, four linear functions in the form of $y = ax + b$ are selected for the model. Another interesting aspect of the measured data is that all three resolvers see very similar correlations between the variables, indicating that the predictive capabilities of the model can be used for other, unobserved resolvers.

7.1.3 Regression

Regression analysis is the process of estimating the relationship between variables. Regression allows estimation of the relationship between a dependant variable and one or more independent variables. The dependent variable is a *regression function* of all the independent variables. The estimated regression function can be used as a predictor for arbitrary inputs of the function. In the parameter estimation process of this thesis, there are essentially four univariate regression functions needed, one for each parameter. The measurements showed that for all four parameters, a linear regression function is sufficient. We have considered two regression models: Simple linear regression (or least square regression) and the Theil-Sen estimator.

Table 7.1: Parameters in the DNS model

Parameter	Details
$\bar{r}$	The average number of responses per outgoing query
α_s	The fraction of responses containing signatures
$\bar{s}$	The average number of signatures per signed response
α_v	The fraction of signatures being validated

Table 7.2: Variables in the DNS model

Variable		Computed by
Responses	R	$R = \bar{r}Q + \beta_1$
Signed responses	R_s	$R_s = \alpha_s R + \beta_2$
Signatures	S	$S = \bar{s}R_s + \beta_3$
Signature validations	S_v	$S_v = \alpha_v S + \beta_4$

Simple linear regression In simple linear regression (SLR), a straight line is fitted on the data, such that the sum of the squared residuals of the model is as small as possible. The residual here is defined as the vertical distance between the fitted line and a data point. While this method finds the smallest error, its major downside is that simple linear regression is susceptible to outliers.

Theil-Sen estimator The Theil-Sen estimator is an alternative for simple linear regression. Similarly to simple linear regression, the Theil-Sen estimator is a straight line that fits the data, but it is computed differently. For every two data points a line through both point is calculated. The median slope of all these lines is the slope of the Theil-Sen estimator. The *intercept* of the Theil-Sen estimator (i.e. the point where the line crosses the point $x = 0$) is also the median of all intercepts. The advantage of the Theil-Sen estimator over simple linear regression is that it is robust (i.e. it is not sensitive to outliers) [53, 54].

Choosing either simple linear regression or the Theil-Sen estimator is a matter of in- or excluding outliers. In our case, predicting the normal behaviour correctly is considered to be more important than modelling the extreme outliers and therefore the Theil-Sen estimator was used in the remainder as the preferred regression method.

7.1.4 Results

The results from the regression methods are four linear functions. The linear functions have a slope and an intersect with the y-axis at $x = 0$. Rather than referring to these slopes by the parameter name a, the parameters are referred to as is shown in Tab. 7.1. In addition, the five variables and their relation are shown in Tab. 7.2. The actual values of the parameters are shown in Tab. 7.3. A visual representation of the fitted lines is shown in Appendix B. As expected, the slopes of the linear functions among the resolvers are fairly similar. Based on the results, the following conclusions can be drawn:

- For every query, the resolvers receive roughly between 0.90 and 0.91 responses ($\bar{r}$).
- On average, between 17.2% and 21.5% of the received responses contain signatures.
- Each signed response contains on average 2.19 signatures.
- On average, between 17.6% and 18.8% of the received signatures is validated.

Table 7.3: The result of the parameter estimation.

Resolver	$\bar{r}$	β_1	α_s	β_2	$\bar{s}$	β_3	α_v	β_4
AMS	0.912	3.714	0.215	-18.284	2.194	6.403	0.188	-1.992
TIL	0.905	2.794	0.187	-7.931	2.163	4.572	0.186	-2.059
UTR	0.904	4.905	0.172	-9.473	2.186	4.067	0.176	-1.318

By combining the four functions from Tab. 7.2, a single, linear function can be defined that has Q as input, and S_v as output:

$$\begin{aligned} S_v &= \alpha_v S + \beta_4 \\ &= \alpha_v(\bar{s}R_s + \beta_3) + \beta_4 \\ &= \alpha_v\bar{s}R_s + \alpha_v\beta_3 + \beta_4 \\ &= \alpha_v\bar{s}(\alpha_s R + \beta_2) + \alpha_v\beta_3 + \beta_4 \\ &= \alpha_v\bar{s}\alpha_s R + \alpha_v\bar{s}\beta_2 + \alpha_v\beta_3 + \beta_4 \\ &= \alpha_v\bar{s}\alpha_s(\bar{r}Q + \beta_1) + \alpha_v\bar{s}\beta_2 + \alpha_v\beta_3 + \beta_4 \\ &= \alpha_v\bar{s}\alpha_s\bar{r}Q + \alpha_v\bar{s}\alpha_s\beta_1 + \alpha_v\bar{s}\beta_2 + \alpha_v\beta_3 + \beta_4 \\ &= \underline{\alpha_v\bar{s}\alpha_s\bar{r}}Q + \underline{\alpha_v(\bar{s}(\alpha_s\beta_1 + \beta_2) + \beta_3) + \beta_4} \end{aligned} \tag{7.1}$$

By replacing the two underlined (constant) terms by a and b respectively, the following equation holds:

$$S_v = aQ + b \tag{7.2}$$

where $a = \alpha_v\bar{s}\alpha_s\bar{r}$ and $b = \alpha_v(\bar{s}(\alpha_s\beta_1 + \beta_2) + \beta_3) + \beta_4$. This indicates that the slope of the curve is defined as the product of all curve slopes for the four individual linear functions. The intercept of the resulting equation is not only a function of the underlying intercepts, but also α_s, $\bar{s}$ and α_v. In other words, if the fraction of responses being signed, the average number of signatures per response or the fraction of signatures being validated changes, not only the slope of the predictor changes, but also the intercept.

Given the measured parameters from Tab. 7.3 and Eq. 7.1, the resulting regression functions that maps outgoing queries to signature validations are as follows:

$$\begin{aligned} \text{AMS} &: S_v = 0.081 \times Q - 8.027 \\ TIL &: S_v = 0.068 \times Q - 4.190 \\ UTR &: S_v = 0.060 \times Q - 3.928 \end{aligned} \tag{7.3}$$

7.2 Model validation

Before the parametrized model was usable for any predictions, its predictive quality had to be validated. The model was parametrized on a data set, but this does not necessarily guarantee that other data would be similar and thus that the model would predict the number of signature validations correctly. The parameters of the model have been determined using linear regression on a one week data set. Our prediction model is not exactly perfect: there is always a residual

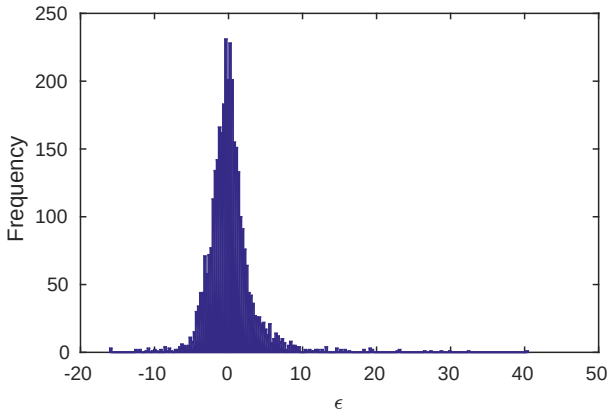


Figure 7.4: Histogram of ϵ for AMS ($\mu = 0.1885$, $\sigma = 3.2746$)

between the predicted value and the actual number of validations, denoted as ϵ. This means that the actual equation for S_v is as follows:

$$S_v = aQ + b + \epsilon \tag{7.4}$$

From our regression process, the value of ϵ is drawn from a random variable. The true probability distribution of this variable is unknown, but given our sample from the regression process, we can estimate the underlying probability distribution. We consider our model to be a good predictor of the number of signature validations if (i.) the mean of the variable approximates 0, (ii.) the variation of the variable is small and (iii.) the distribution of the variable is equal to the distribution of a separate validation data set. The first two requirement can be met by simply observing a histogram of ϵ, as displayed in Fig. 7.4. The mean is sufficiently close to zero (i.e. 0.19) and the standard deviation is sufficiently small (i.e. 3.27). The remaining issue is to prove that the error in the prediction is equal among different data sets. This has been validated using the Kolmogorov-Smirnov test.

7.2.1 Kolmogorov-Smirnov test

According to [55], the Kolmogorov-Smirnov test (or KS test) can be used to test whether a particular distribution $\mathbb{P}$ is equal to a given specified distribution $\mathbb{P}_0$. For example, $\mathbb{P}$ could be the distribution of height in men, where $\mathbb{P}_0$ is a normal distribution. In terms of hypotheses, the KS test can decide between the following hypotheses:

$$\begin{aligned} H_0 &: \mathbb{P} = \mathbb{P}_0 \\ H_1 &: \mathbb{P} \neq \mathbb{P}_0 \end{aligned} \tag{7.5}$$

If the Kolmogorov-Smirnov test accepts H_0, both distribution are assumed to be equal. In our case, $\mathbb{P}$ would be the distribution of the residual term ϵ, but $\mathbb{P}_0$ would not be a given specified distribution. Namely, $\mathbb{P}$ would be the distribution of the residual term of the *validation* data set. Therefore, a two-sample KS test had to be applied.

The two-sample KS test requires two samples with distributions F and G, which are of size m and n respectively. The empirical cumulative distribution functions (ECDFs) are denoted as

$F_n(x)$ and $G_m(x)$. The KS statistic for a two-sample test is defined as follows:

$$D_{mn} = \sup |F_m(x) - G_n(x)| \tag{7.6}$$

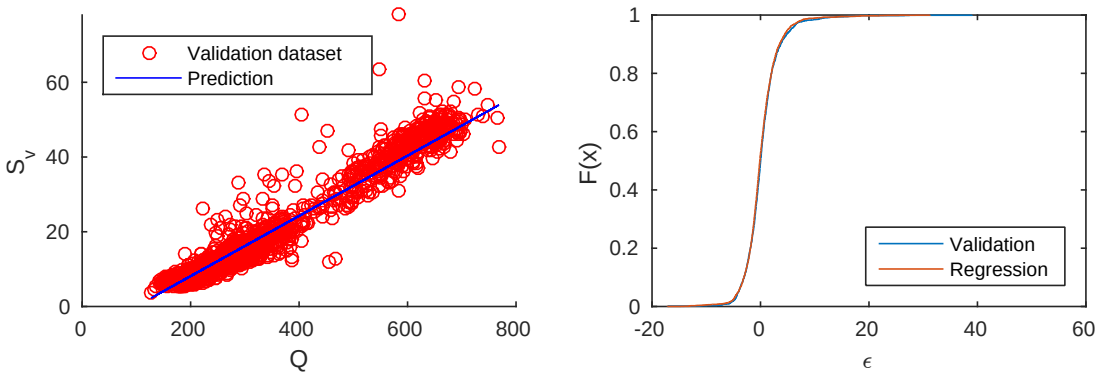
where sup is the supremum (or maximum) function. In other words, the KS statistic is the maximum distance between the two ECDFs. The null hypothesis is rejected when $D_{mn} > c(\alpha)\sqrt{(n+m)/(nm)}$, where α is the significance level and $c(\alpha)$ can be obtained from a critical value table [56]. The critical value table from [56] states the critical value for a one-sample Kolmogorov-Smirnov test and this value is adjusted for the two-sample test that we use. In other words, if the KS statistic exceeds the critical value $c(\alpha)\sqrt{(n+m)/(nm)}$, the two samples are from different distributions .

7.2.2 Results

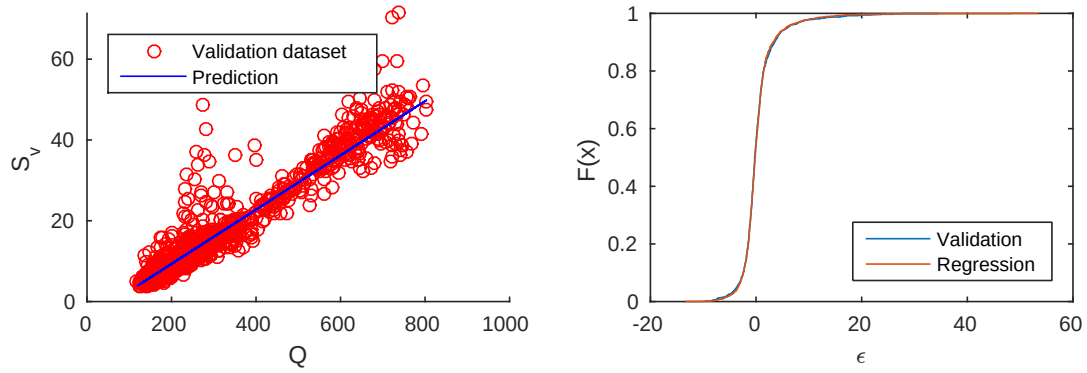
In order to apply the two-sample Kolmogorov-Smirnov test, we needed two different sets of ϵ samples. The first set was extracted using the regression session, and is denoted as $\mathbb{R}$. In reality, not the full one week data set was used during the regression phase. We separated the 5,040 data points in the regression set (70%) and a validation set (30%), where only the first set was use for regression. The data points were selected by randomly selecting 70% of the full week data set. This ensured that different points in time would incorporated (e.g. weekend days, nightly hours, etc.). The remaining data points were used for the validation set. All residual terms were obtained by subtracting the predicted value from the actual number of validations. The distribution for the validation set residual terms is denoted as $\mathbb{V}$. The two-sample Kolmogorov-Smirnov test was applied to determine whether the probability distribution of $\mathbb{R}$ and $\mathbb{V}$ are significantly different or not. We decided that a significance level of 5% would be sufficient. Namely, this value is common in scientific research as it provides a good balance between type I (i.e. rejecting the null hypothesis while it is true) and type II errors (i.e. accepting the null hypothesis while it is false). The hypotheses are as follows:

$$\begin{aligned} H_0 &: \mathbb{R} = \mathbb{V} \\ H_1 &: \mathbb{R} \neq \mathbb{V} \end{aligned} \tag{7.7}$$

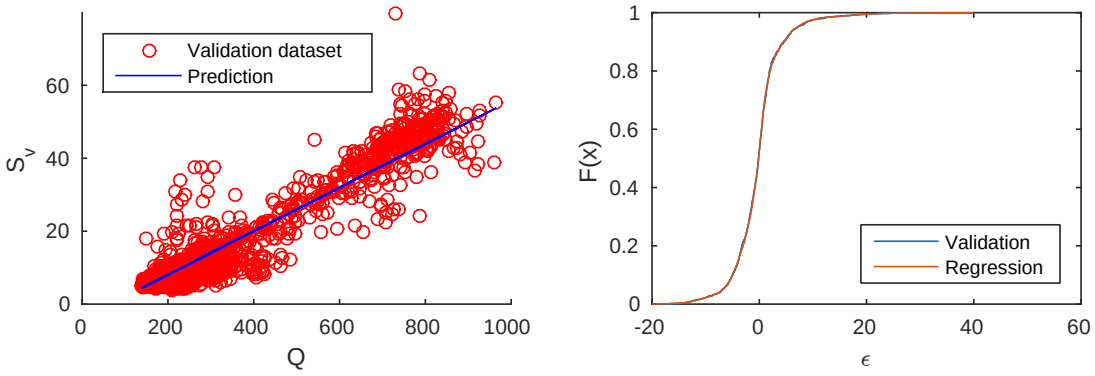
The left three plots in Fig. 7.5 shows the prediction (blue line) for the red validation data points. The line seems to be a good fit, which is confirmed by the ECDF of the right three plots. The red line is the ECDF of the regression data set and the blue line is the ECDF of the validation data set. For all three resolvers, the ECDFs are very similar to each other indicating that the Kolmogorov-Smirnov test may accept the null hypothesis. The critical value for a significance level of 5% is $1.36\sqrt{\frac{5040}{1512*3528}} = 0.042$. The KS statistic obtained from the ECDFs are 0.034, 0.020 and 0.016 for AMS, TIL and UTR respectively. Since all these value lie below the critical value, the null hypothesis for the three resolvers is accepted (i.e. the regression and validation data set come from the same distribution) and therefore our model is deemed to have sufficient predictive capabilities.



(a) AMS



(b) TIL



(c) UTR

Figure 7.5: **Left:** the predicted values for the validation data set, **Right:** ECDFs for the two-sample KS test.

Chapter 8

Scenario evaluation

Now that the DNSSEC model is validated against a different data set, we can claim that our model is suitable for predicting future scenarios. In the next sections we discuss two sets of relevant scenarios: the current state of DNSSEC and future scenarios where the DNSSEC deployment is more prevalent. In these scenarios, the predicted number of signature validations should be lower than the maximum number of validations that a resolver can support (i.e. the validation threshold). In the next sections, the approach to finding the validation threshold is described and afterwards the scenarios are evaluated.

8.1 CPU benchmark

Assuming that the CPU load caused by validations is the most dominant factor in the total CPU load of a resolver, the predicted number of validations should lie below a threshold. The value of the threshold (i.e. maximum possible number of signature validations per second) depends on the cryptographic algorithm, where we can already assume that the threshold of ECDSA P-384 lies lower than any other investigated algorithm, in particular RSA -1024. The exact values of the threshold were obtained in a benchmark experiment. The benchmark was used to measure the CPU utilization for not only a single signature validation, but also to verify if incrementing the number of signature validations will result in a linear increase of CPU load. The benchmark had the following features:

- A computer system with similar specifications as the SURFnet resolvers performed signature validations. This system contained a 2.27 GHz dual-core processor. The benchmark was performed on a single core. In the benchmark, the validations were performed on a single pre-generated signature.
- In order to verify whether the CPU scales linearly with the number of validations per second, the benchmark was executed for different frequencies of signature validations (i.e. iterations), ranging from almost none to a the number of validations resulting in 100% CPU usage.
- Each iteration ran for five seconds, where the CPU load was measured as follows:

$$\text{CPU load} = \frac{\text{validation CPU usage}}{\text{total system CPU usage}}$$

  The CPU load was measured over a five second period because the CPU could be measured with 10 ms accuracy, resulting in a maximum of 0.2% error which was deemed acceptable.

- The benchmark was performed for five cryptographic algorithms: RSA-1024, RSA-2048, ECDSA P-256, ECDSA P-384 and Ed25519. The first two algorithms are currently used widespread, while the two ECDSA algorithms are the two most likely replacement algorithms. Ed25519 provides the same benefits as ECDSA and in addition is claimed to be much faster than ECDSA. This means that in theory, the algorithm has all the advantages of both RSA and ECC.
- For the RSA benchmarks, the used hash function is SHA-1. The performance difference between SHA-1 and SHA-2 is considered to be negligible. For ECDSA there exists no choice in hash function: ECDSA P-256 used SHA-256 and ECDSA P-384 used SHA-384.
- The validations were executed using the OpenSSL library for C. OpenSSL was selected because Unbound uses the same library for its signature validations. The benchmark program was written in such a manner that all possible variables were eliminated during the test, such that the CPU load was measured as precisely as possible. In case of Ed25519, a different implementation was selected, since the algorithm is not implemented in OpenSSL (yet). The used Ed25519 library[1] internally uses the original Ed25519 implementation from its authors (named 'ref10') and unfortunately the used implementation is somewhat slower than 'ref10'. Since the benchmark was performed with different implementations, the results are not directly compared with each other but merely provided as an indication. The different implementations within OpenSSL are compared with each other.
- The benchmark was performed for three different versions of OpenSSL: version 1.0.0, version 1.0.1f and version 1.0.2d. The first one is an early implementation, the second implementation is the default version on Ubuntu 14.04 LTS and the latter is the newest OpenSSL version. By performing the test on different versions we hoped to observe an increase in performance over the different implementations, indicating that the algorithms are being implemented more efficiently.

The implementation of the benchmark can be found in Appendix C. Since the benchmark test is performed on a single core, the results are not representative for the complete system, because systems generally have multiple cores and thus the total capacity might be two, four or more times as large. We provide the measurements for a single core and potential extrapolation for multiple cores is deemed outside the scope of our research.

8.1.1 Assumptions

When evaluating the performance limits of a validating DNS resolver, several assumptions have been made. The most important are the following:

1. In the discussion of the performance of a resolver, we have only considered the CPU utilization as a bottleneck. The validation of a signature essentially consists of the multiplication and addition of large integers, which takes some computation time and just a little memory. In addition, when performing more validations, the memory usage does not increase since validations are generally performed sequential rather than in parallel. Therefore, increasing the number of validations will mainly increase the CPU usage of the resolver rather than its memory (or other resources).
2. The majority of the total usage of CPU will be dominated by signature validations. Fig. 8.1 shows the expected CPU usage of a resolver when the number of validations increases. Initially, with no validations, the resolver consumes a small percentage of the available computation power, the base CPU usage. This base CPU time is spent at default DNS

[1] https://github.com/orlp/ed25519

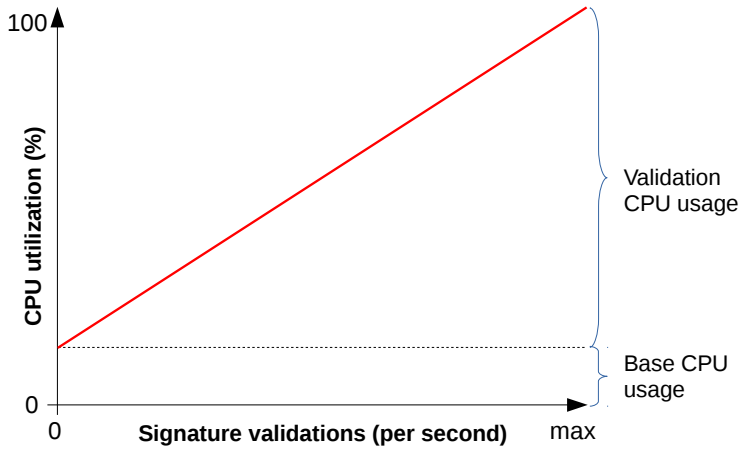


Figure 8.1: The expected development of CPU usage for different amounts of signature validations.

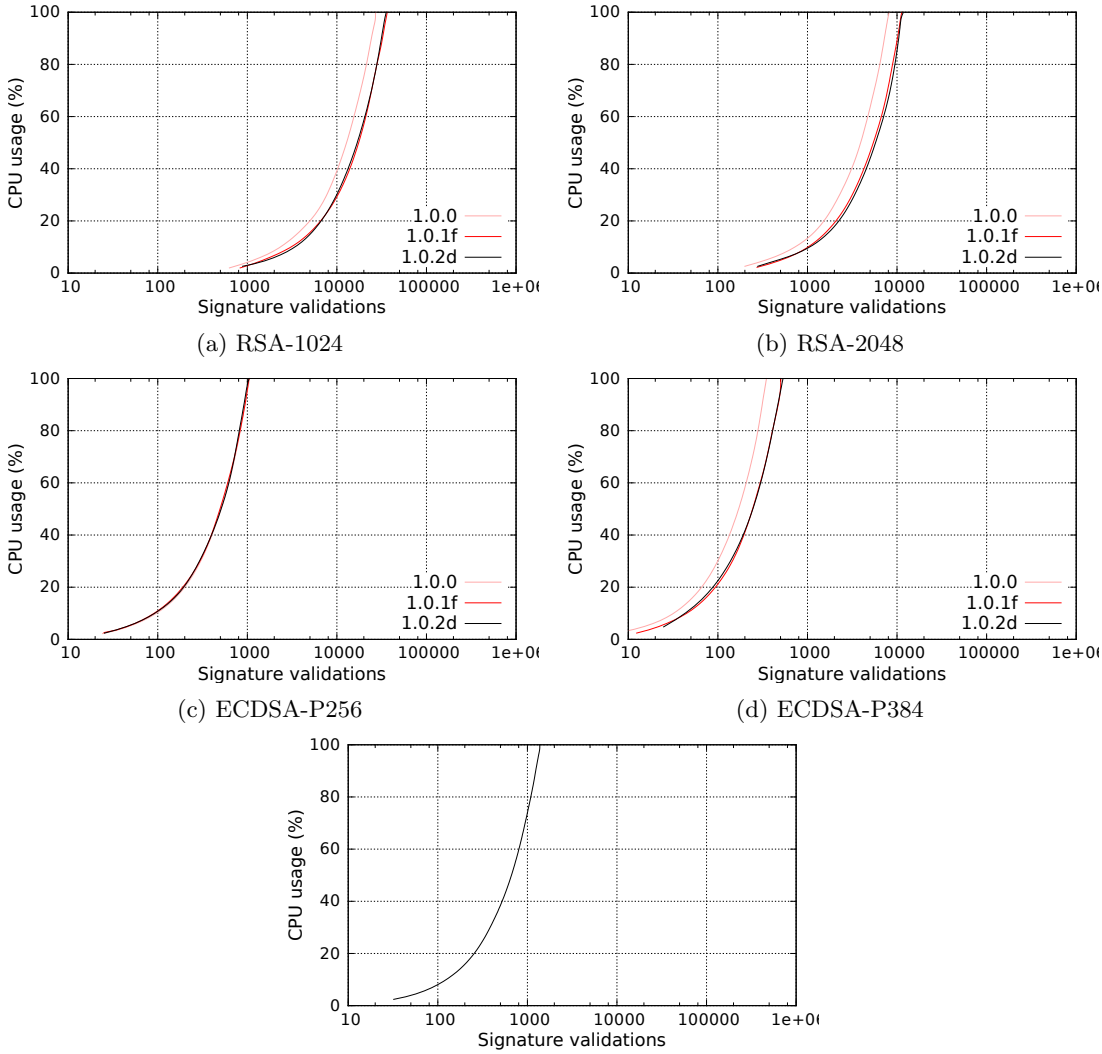
resolver behaviour (e.g. answering answers from the cache, writing to the cache, performing recursive lookups) and other running processes. The fraction of the CPU usage consumed due to signature validations is referred to as the 'validation CPU usage'. At the maximum possible validations, the base CPU usage is considered to be negligible compared to the validation CPU usage. Whether this assumption holds in reality is unknown, but falls outside the scope of this research. We believe that a sufficient accurate prediction of the maximum amount of signature validations can be made regardless.

8.1.2 Results

Fig. 8.2 shows the results from the CPU benchmark. The sub figures each show one of the five algorithms. Within each plot, the three versions of OpenSSL are displayed, with the exception of the last plot (Ed25519) since the algorithm is not implemented in OpenSSL. Note that the x-axis is logarithmic rather than linear. From the figures the (expected) linear trend cannot be seen, but we have indeed confirmed that a linear relation exists between the number of validations and the CPU load. As expected, RSA-1024 is capable of the most signature validations per second, followed by RSA-2048, Ed25519, ECDSA P-256 and lastly ECDSA P-384. Even the supposedly sub-optimal implementation of Ed25519 is faster than both ECDSA algorithms, which confirms the previous claim that the algorithm is a suitable, faster alternative to ECDSA.

A significant difference can be observed between the different oldest version of OpenSSL (1.0.0) and the two new versions (1.0.1f and 1.0.2d). For RSA-1024, RSA-2048 and ECDSA P-384, the maximum signature validations increased with a factor of over 1.3, where ECDSA P-256 only showed a small increase in performance. Almost no difference can be observed between version 1.01f and 1.0.2d which may indicate that the performance increase of the cryptographic algorithms has not been a priority, or that a performance cap has already been reached.

The data (summarized in Tab. 8.1) shows that SURFnet's DNS resolvers can perform 38,505 signature validations with RSA-1024, 12,145 with RSA-2048, 1,452 with Ed25519, 1,125 with ECDSA P-256 and only 565 with ECDSA P-384 per second, assuming that OpenSSL version 1.0.1f is being used. For the remainder of the report, we assume that a system used this version,



(a) RSA-1024

(b) RSA-2048

(c) ECDSA-P256

(d) ECDSA-P384

(e) Ed25519

Figure 8.2: The results of the CPU benchmark

Table 8.1: Maximum achieved validations per second for different versions of OpenSSL.

Algorithm	Maximum validations (per second)		
	v1.0.0	v1.0.1f	v1.0.2d
RSA-1024	28,555	38,505	37,236
RSA-2048	8571	12,145	12,461
ECDSA P-256	1070	1125	1090
ECDSA P-384	358	527	565
EdDSA	1452		

Table 8.2: Distribution of signature validations among categories.

DNS hierarchy	Validation key	Percentage of validations
Root	KSK	0.0002%
	ZSK	1.05%
TLD	KSK	0.14%
	ZSK	44.15%
SLD and lower	KSK	6.81%
	ZSK	47.84%

because it is the default version of OpenSSL. These numbers can now be used as a threshold in the scenario evaluation process. If the predicted number of validations fall below the aforementioned threshold, the DNS resolver is capable of serving all its clients. If the number of validations falls above the threshold however, a hardware upgrade is required.

8.2 Current scenario

The first scenario that was analysed is a current state scenario, where all RSA signatures are replaced by ECDSA P-256, ECDSA P-384 or EdDSA signatures. In particular, the worst case scenario is of interest. Namely, if a validating DNS resolver is able to cope with the increased CPU load caused by ECC, then an overnight shift towards ECC would cause no problems. The worst case scenario is simply defined as the highest number of signature validations that has been encountered by a resolver. This number was acquired by measuring the number of validations over a period of one week. Since AMS, TIL and UTR have the same system specifications, the worst case scenario is shared between them. During the one week measurement, the highest number of signature validations was 132.5 validations per second. Note that this was an average of 120 seconds and thus the actual number of validations may have peaked higher than this number.

According to Tab. 8.1, the most computation heavy algorithm, ECDSA P-384, is capable of validating 527 signatures per second, given that all other factors on the computation load (e.g. processing cache hits or running the operating system) are negligible. Clearly, the current worst case scenario of 132.5 validations does not even come close to the maximum possible number of signature validations for ECDSA P-384. ECDSA P-256 is even more favourable, because it is capable of performing 1,125 validations per second.

8.2.1 Introducing the combined signing key

Earlier in the report, the advantages of ECC over RSA combined could potentially result in a situation where the KSK and ZSK mechanism could be replaced by a single combined signing key (CSK). According to Van Rijswijk et al. [52], it is unlikely that high level domains, such as the root and the TLDs will drop their KSK and ZSK mechanism in favour of a single CSK, due to security reasons. In contrast, lower level domains (i.e. SLDs and lower) may all employ a CSK. This means that in the future, no more signature validations using the KSK at the second level or below will be performed any more.

Tab. 8.2 shows the distribution of signature validation using either the KSK or ZSK, and using either keys from the root/TLDs or lower tiers. All signature validations on the second tier and below using the KSK will longer occur when ECC is used and thus (according to the table) 6.81% less validations are performed. In our current worst case, this would mean that approximately 9 out of the 132.5 validations per second would not need to be performed any more. The already favourable current scenario becomes even more favourable by adopting the CSK.

In conclusion, the current state of DNSSEC poses no problem to a general validating DNS resolver in terms of computation power, so an overnight shift towards any of the standardized ECC algorithms or Ed25519 can be accomplished without any CPU related issues. The question remains whether we can draw a similar conclusion for future scenarios.

8.3 Future scenarios

We identified two factors that will likely impact the number of signature validations in the future. Firstly, the total number of domains on the Internet may increase, which causes an arbitrary resolver to perform queries for more and different domains. This causes more signature validations to take place. Secondly, the DNSSEC deployment is likely to increase. Since the root has been signed in 2010, the number of signed TLDs has been increasing steadily and with it also the number of domains. It is not unthinkable that in the near future, a majority of the domains are signed. The growth in total number of domains in DNS is relatively slow compared to the growth of DNSSEC deployment. Therefore, when considering a future scenario, a DNSSEC deployment increase is important to consider.

The scenario of increased DNSSEC deployment can be evaluated with our model. The DNSSEC deployment is not included as parameter in the model, but it directly influences one of the parameters: the fraction of responses containing signatures. The fraction of responses containing signatures (or α_s) can be estimated using the DNSSEC deployment combined with the popularity of the domain names. Namely, an unpopular domain is rarely requested by resolver and by signing that domain, the number of responses with signatures will hardly increase. The opposite is true for a popular domain: by signing it a significant amount of responses may contain signature as a result of signing a single domain. As a result, an estimation of parameter α_s can be made based on the DNSSEC deployment and the popularity of the signed domain names.

Unfortunately we have no access to historical data regarding the growth of DNSSEC and the fraction of responses containing signatures. The current situation states that between 1.685 and 3% (see Section 6.3.2) of domains is signed, while the value of α_s lies between 17.54 and 18.75%. We assume that the growth of α_s at least lies between an upper and lower bound, defined by the worst case scenario and a uniform scenario. In the worst case scenario, the most popular

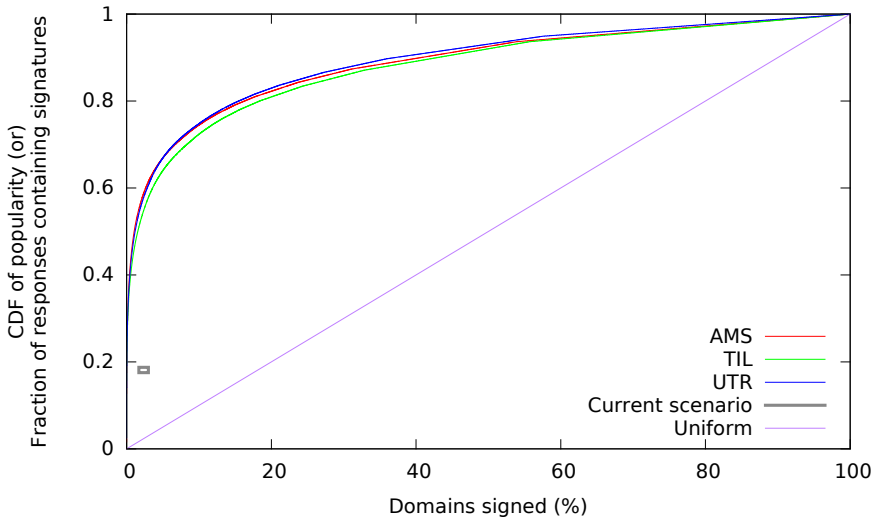


Figure 8.3: The cumulative distribution function of domain name popularity in outgoing queries.

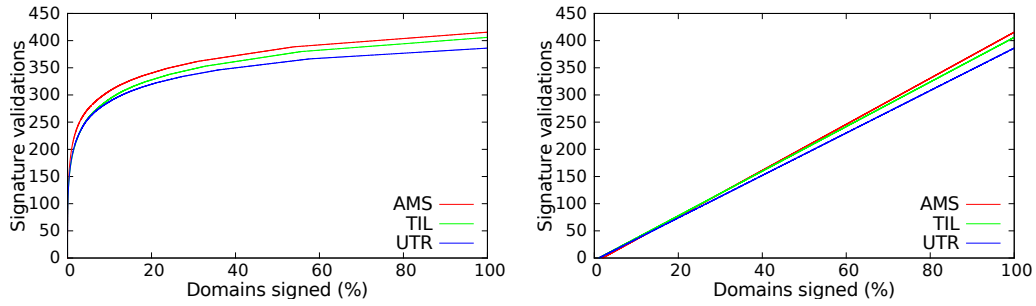
domain name is signed first, the second most popular domain name is signed secondly, etc. This indicates that whenever 5% of all domains is signed, the 5% most popular domains are signed. If the top 5% of the most popular domain names are signed, we can expect that the CDF of domain name popularity up to 5% determines the fraction of responses containing signatures. In other words, the CDF of the domain name popularity describes the worst case growth of α_s. In a uniform growth scenario, the domains are signed uniformly. This means that on average, α_s is equal to the DNSSEC deployment. Fig. 8.3 displays a visual representation of the upper and lower bound. The gray area in the bottom left of the screen represents the current scenario. The uniform scenario is the straight purple line. The worst case scenario is displayed for the three different resolvers, because the domain name popularity differs slightly for the three resolvers. The figure displays that the current scenario indeed lies between the upper and lower bound.

In the following sections, the growth for different DNSSEC deployment percentages is evaluated for the worst and the best case scenario.

8.3.1 DNSSEC deployment growth

As already briefly discussed in the previous section, the red, blue and green lines in Fig. 8.3 represent the value of α_s for different percentages of DNSSEC deployment for AMS, TIL and UTR respectively. The domain name popularity distribution was obtained from the data in our previously performed one week measurement. The datasets did not contain all global domain names, simply because the majority of these domain names was never requested by the clients of any of the resolvers. The datasets (of the three resolvers) contained between 2.53 and 3.01 million domain names, which is estimated to be 0.86 to 1.02% of all domains worldwide [57]. We have normalized the data such that our dataset represents 100% of the domains so that we can still provide a 0-100% DNSSEC deployment scenario (rather than a 0-1.02% scenario).

Given a new parameter α_s obtained from Fig. 8.3, the number of signature validations (S_v) can be computed from the input (i.e. the outgoing queries or Q). Fig. 8.4a shows the number



(a) Worst case

(b) Uniform

Figure 8.4: Number of signature validations for $Q = 1122.22$ with different DNSSEC deployment values.

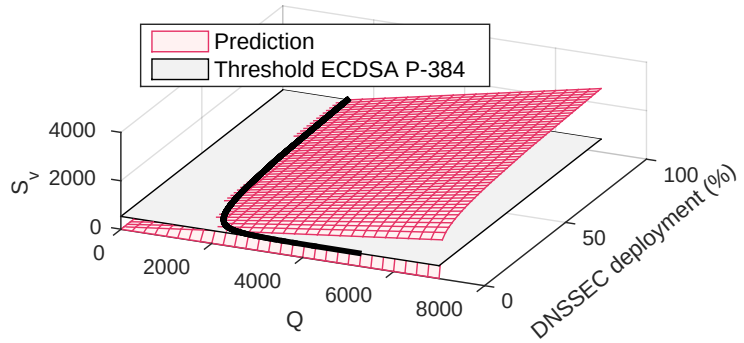
of signature validations given the highest number of Q that was measured on any of the three SURFnet resolvers, namely 1122.22 outgoing queries per second. The figure shows that even in the absolute worst case, where every single domain in the world is signed, between 387.21 (UTR) and 414.46 (AMS) signature validations are performed, still less than the threshold of 527 for ECDSA P-384. Thus even in the worst situation (with 1122.22 outgoing queries and a DNSSEC deployment percentage of 100%), the number of signature validations remains acceptable.

For full DNSSEC deployment our single-core test resolver should be able to cope with our current worst case scenario. Therefore, it does not matter what curve the value of α_s has since for 100% DNSSEC deployment the values of any curve would result in the same amount of validations. For completeness purposes, the number of validations for the uniform case are shown in Fig. 8.4b. Note that due to errors in the linear regression the three lines do not intersect the point $(0, 0)$.

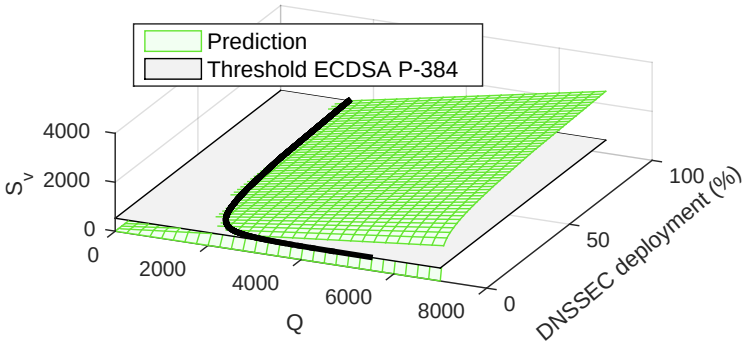
8.3.2 Increased outgoing queries

Increasing the DNSSEC deployment is not the only way that the number of signature validations may increase in the future: the number of outgoing queries may also increase. It is likely that the number of outgoing queries for an arbitrary resolver will only grow in the future, given that more domain names are registered which is the main cause for an increase in outgoing queries. Due to the caching mechanism in a resolver, incoming queries for already popular domain names will (hardly) increase the computation load for a resolver, because the query is immediately answered from the cache. The growth in the number of outgoing queries is expected to be fairly small compared to the DNSSEC deployment, mostly because the DNSSEC deployment is very low and has been growing largely since the zone was signed in 2010. Nevertheless, analysing how the growth of the outgoing queries impacts the number of signature validations is still an interesting scenario for the future of DNSSEC.

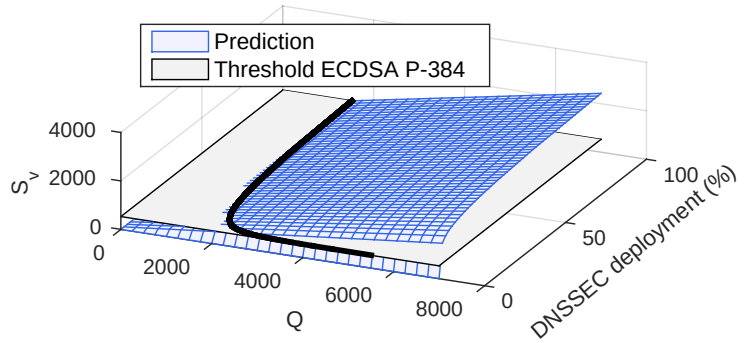
Our model can be used to calculate the value of Q that results in the threshold value of 527 ECDSA P-384 validations. In Fig. 8.5, the estimated number of signature validations using the model is displayed by the red, green and blue surfaces. The gray plane shows the maximum possible amount of signature validations for ECDSA P-384. The intersection of these two planes (in black) shows the maximum value of Q is supported by each validating resolver. At 100% DNSSEC deployment, the maximum supported outgoing queries is the least, since



(a) AMS



(b) TIL



(c) UTR

Figure 8.5: Number of signature validations for DNSSEC deployment

per outgoing query the most amount of validations need to be performed. This value of Q lies between 1422.6 and 1525.6 for AMS and UTR respectively. This means that the volume of outgoing queries needs to grow by a factor of 1.27 to become a problem, and that only happens when every single domain name is signed with ECDSA P-384. In cases where the DNSSEC deployment is less than 100%, even more outgoing queries are possible, reducing the problem even more. The same approach has been used to compute the tipping point for ECDSA P-256 where a resolver is not able to cope with the signature validation load any more. Since validating a single ECDSA P-256 signature is less computation intensive, a resolver is able to validate more signatures per second compared to ECDSA P-384. According to Tab. 8.1 from the previous chapter, a total of 1,125 signatures can be validated per second with ECDSA P-256. In case of 100% percent DNSSEC deployment, the outgoing query traffic needs to grow with a factor 2.69 (to a total of 3015.13 outgoing queries per second) to reach the maximum CPU utilization.

For lower DNSSEC deployment values, Q needs to grow even more before the CPU load becomes a problem. The current estimates of the DNSSEC deployment range from 1.685% to 3%. Consider an example where the deployment increases to 6%: a doubling of the highest current estimate but still far lower than full DNSSEC deployment. In Fig. 8.5, the intersection (i.e. the black line) of the prediction plane and the threshold plane indicates the maximum supported number of outgoing queries for that particular percentage of DNSSEC deployment. By finding the point of the black line for the DNSSEC deployment percentage of 6%, we can see that the maximum supported outgoing queries is 2,068. Thus, an increase of a factor 1.84 of the current highest number of outgoing queries would result in a problematic situation when 6% of all domains are signed. A situation where 6% of all domains is signed may be very well possible in the near future, but an increase of the outgoing queries by a factor 1.84 is far more unlikely.

In conclusion, given the current maximum number of outgoing queries there is still a little growth left regarding the number of outgoing queries even when there is full, global DNSSEC deployment. Our model can be used to evaluate intermediate situations, such as situations with partial DNSSEC deployment.

Chapter 9

Discussion

While we have been able to demonstrate that our model performs well, several key issues and observations still need to be discussed. These points of discussion relate to the modelling as well as the measurement aspects of the research.

Data outliers The measurements have shown that resolvers are exposed to heavy varying amounts of traffic and sometimes experience extreme peaks of traffic. These outliers in the data seem to occur randomly. For example, consider Fig. 6.3, which displays the incoming and outgoing queries for the three resolvers. The large peaks in the incoming queries are caused by changes in the behaviour of clients. Causes may include attempts to DDoS attacks, a sudden increase of client population or an increase in activity per person.

Several peaks can also be observed in the outgoing query traffic. For example, the AMS resolvers in Fig. 6.3b on page 40 shows a peak on Thursday night in its outgoing queries. This (and similar) peak(s) could be cause by several reasons:

- An outgoing query is generated when an incoming query is received and the query is not cached. Therefore, a peak in the outgoing queries may be generated when the TTL of many cache entries expire simultaneously. In our experimental measurements (Chapter 6) the TTL of RR sets often have the same value (e.g. 5 minutes, one hour, one day, etc.). If these RR sets were cached at the same time (for example right after a reboot of the resolver) these RR sets may indeed expire at the same time. However, this is unlikely to happen at such a large scale, since the RR sets are not immediately cached if the TTL expires (i.e. the RR set should be requested first by the client.
- If the client population radically changes the domains they request (i.e domain names that are not cached) or suddenly diversify the requested domain names, a sudden surge of outgoing queries may be generated.

The latter explanation is more likely than the first one. However, this does not yet explain why the clients suddenly diversified their query traffic. The exact reason can only be guessed and may be dependant on many factors.

DNSSEC deployment In Chapter 8, two cases were evaluated: the worst case scenario and the uniform scenario. These scenarios are far from likely to be realistic, but give an upper and lower bound to the growth of the fraction of responses containing signatures (α_s) given the DNSSEC deployment. In reality, the DNSSEC deployment may proceed very differently. A few

influences on the DNSSEC deployment are discussed below.

In contrast to the incoming domain popularity, the outgoing popularity does not reflect the actual popularity from clients. Namely, popular domain names at the client side are generally answered from the cache and the domain name will only appear in the outgoing query traffic when its TTL expires. When Unbound is not able to resolve a domain name, it will aggressively retry and try other name servers. In case of misconfigured zones, the domain name may not be resolved at all while Unbound still generated lots of traffic for that domain. This means that a majority of the most popular outgoing query names are from misconfigured zones. These misconfigured zones are unlikely to introduce DNSSEC at all, let alone introduce it before all other domains.

Additionally, in the scenario evaluation the worst case specifies that domains are signed one by one. In reality, it is more likely that large groups of domains (i.e. operated by a single service provider) will be signed rather than a single domain within a zone. Some of the domains in the zones may be very popular while others may not. For example, consider a zone with a popular web server (`www`) and a rarely used mail server (`mx`) in the same zone. These domains lie on a very different place on the domain popularity curve but are still (presumably) signed simultaneously.

Lastly, the DNSSEC deployment may be heavily stimulated on a TLD level which causes domains within a particular TLD to be signed in quick succession. For example, within the `.nl` TLD, 43.7% of all domains is already signed while within `.com` only 0.45% of all domain is signed[1]. Thus, DNSSEC deployment is presumably not (heavily) dependant on domain name popularity, but rather whether the parent TLD of a domain actively encourages deployment of DNSSEC.

CPU load In Sec. 8.1, the different versions of OpenSSL already showed that there was room for improvement in the implementation of cryptographic algorithms. There was a major improvement between version 1.0.0 (from March 2010) and version 1.0.1f (from January 2014) and compared to RSA, ECDSA is a recently developed algorithm. Therefore, it is likely that more improvement is possible. The case for Ed25519 is even more favourable. The main implementation of the protocol is a proof-of-concept from its inventors, which indicates that the development of the algorithm is in its infancy and has much room for improvement. Therefore, the performance of ECC algorithms may become relatively better in the near future which makes the deployment of ECC instead of RSA even more favourable.

[1] http://rick.eng.br/dnssecstat/

Chapter 10

Conclusions and Future Work

In our research we have proposed a DNSSEC model that is capable of predicting the number of signature validations of a DNS resolver. Using the proposed model in combination with DNS traffic measurements, we were able to answer all the research questions formulated towards our research goal.

What is the typical computation load of a DNS resolver?
The traffic of three DNS resolvers (denoted as AMS, TIL and UTR) has been investigated during the course of the project. Typically, the number of signature validations on these resolver lies between almost none to 132.5 validations per second. Currently, the vast majority of all signature validations are performed with RSA (99.96%). Our CPU benchmark tests have shown that depending on the algorithm and the version of OpenSSL, a single-core CPU is able to verify between 8,571 and 38,505 RSA signatures per second. Since the CPU load scales linearly with the number of signature validations, the CPU load caused by signature validations is between 0.34% and 1.54%. In conclusion, currently the CPU utilization caused by signature validations can be considered to be negligible.

What is the typical behaviour of clients, resolvers and name servers?
In the exploratory measurements, the results showed that resolvers are subjected to varying amounts of traffic. Clients of the SURFnet resolvers generate queries in a day and night cycle where the most queries are generated at midday. In addition, a weekday and weekend cycle was observed. Clients requested a very small subset of domains with a very high frequency. The popularity of incoming domain names can be described using a Zipf-distribution.

In terms of resolver behaviour, there is a difference between the three SURFnet resolvers. AMS is exposed to significantly more queries than the other two, presumably because in a list of resolvers, the AMS resolver is generally the first entry. Regarding outgoing queries, the three resolvers are behaving similarly. The caching behaviour of DNS resolvers causes the difference between incoming and outgoing queries. The complicated relationship between TTL values, popularity of queried domain names and DNSSEC deployment was the reason for omitting the incoming queries as part of the model. Unbound appears to have implemented an aggressive retry mechanism, because a significant fraction of the most popular outgoing query domain names are non-existing domains. This means that there is no (or a very weak) correlation between the popularity of an incoming query and its associated outgoing popularity.

How can we model this behaviour and what are the relevant parameters?
In our research, a prediction model using regression was used, in particular the Theil-Sen estimator. The experimental setup gave an indication of the relevant parameters. The four parameters in the model (average responses per outgoing query or $\bar{r}$, the fraction of responses containing signatures or α_s, the average number of signatures per signed response or $\bar{s}$ and the fraction of signatures being validated or α_v) were chosen because of several reasons. Firstly, the parameters were considered to be independent of each other, meaning that we could change one of them without affecting the others. Secondly, the parameters appeared to be fairly constant over time, meaning that the model is applicable to all moments in time, including day- and night hours and week- and weekend days. Lastly, the parameters were similar among resolver, making the model usable for an arbitrary DNS resolver, rather than just a single specific resolver. By changing only the fraction of responses containing signatures (α_s), future scenarios could be evaluated.

What scenarios can we evaluate to measure the impact of deploying ECC?
The following scenarios were evaluated in our research:

#1 : The current worst case scenario (i.e. the case were most signature validations were performed)

#2 : Future worst case with maximum outgoing queries and varying DNSSEC deployment

#3 : Future worst case with varying outgoing queries and varying DNSSEC deployment

Main research question: What is the effect of deploying ECC as a replacement of RSA in DNSSEC on the computation load of validating DNS resolvers?
If the transition towards ECC would be made with the current state of the DNSSEC infrastructure, no issues regarding the computation load of validating DNS resolvers would be encountered. Even in the worst possible situation (i.e. encountering the maximal realistic number of signature validations per second) the transition to the worst algorithm (in terms of computation load), namely ECDSA P-384, would be possible. According to our regression model, the number of validations could quadruple before becoming a problem.

Whenever a CSK is used rather than a KSK and ZSK, the potential problem is reduced even more. Namely, an estimated 6.81% of all validations (i.e. the validations with the KSK on a SLD level or lower) does not need to take place any more.

In case DNSSEC becomes fully adopted globally, the predicted number of validations still does not exceed the ECDSA P-384 threshold (i.e. the maximum number of signature validations per second that a validating DNS resolver is capable of performing), let alone the ECDSA P-256 threshold. The prediction states that the number of validations does approach the ECDSA P-384 threshold, so resolvers with larger client populations or less powerful resolvers may suffer from CPU starvation. The model also allows to evaluate intermediate scenarios, where the domains are partially signed.

10.1 Future work

Despite having produced proof that the current (and future) state of DNSSEC can be supported by a validating resolver, there still remain some unanswered questions. We discuss the most important remaining issues that should be examined in future research.

BIND resolver
The entire research was conducted using Unbound as resolver software. However, there are other software suites available that implement a validating resolver. The most notable of these alternatives is BIND, which is the most deployed DNS server worldwide. If the research would have been conducted using BIND, the results could have been different. Small implementation differences could have impacted the results. For example, the timeout strategy for both resolver tools could be different. We already saw that the popularity distribution of outgoing query names is dominated by unanswered query names. If BIND does not retransmit as much queries as Unbound does, then the domain name popularity curve could look differently. For a 100% deployment scenario the results would not differ, but the intermediate DNSSEC deployment percentages it would. Additionally, BIND may have different (default) settings regarding caching. For example, Unbound has a (default) maximum TTL value for any RR set, which may not be the case for BIND.

Model validation
The parameter estimation phase showed that among the three SURFnet resolvers there is little difference in parameter values. It seems that the three resolvers share the parameters, but the question remains whether the same holds for non-SURFnet resolvers. The clients of AMS, TIL and UTR are generally educational or research oriented persons, and therefore the traffic has a tendency for particular educational/research domains. Even though other resolvers may see different domains, this does not necessarily imply that our model is inapplicable. More research should show whether the predictive capabilities of the model still holds for different resolvers.

Extended data set
The data set used for the parameter estimation was limited to a one week measurement. The data showed that among day and night hours the traffic already heavily fluctuates and the same holds for week days and weekend days. A more large scale cycle could also be present, which includes for example holidays.

In relation to the day-night cycle and the week-weekend cycle, our model is a single linear regression model, where one outcome (S_v) is based on a single predictor (Q). In the future, the model could extended to include a time related variable to establish a multiple regression model.

Bibliography

[1] P. Hoffman and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC," RFC 6605 (Proposed Standard), Internet Engineering Task Force,

[2] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Y. Yang, "High-Speed High-Security Signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[3] B. Schneier, "Applied cryptography: protocols, algorithms, and source code in C," 1996.

[4] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[5] "The Case for Elliptic Curve Cryptography," National Security Agency, Accessed via: https://www.nsa.gov/business/programs/elliptic_curve.shtml

[6] "How DNS works," Accessed via: https://technet.microsoft.com/en-us/library/cc772774%28v=ws.10%29.aspx#w2k3tr_dns_how_mlkk

[7] P. Mockapetris, "Domain names - Implementation and specification," RFC 1035 (Internet Standard), Internet Engineering Task Force,

[8] P. Vixie, "Extension Mechanisms for DNS (EDNS0)," RFC 2671 (Proposed Standard), Internet Engineering Task Force,

[9] S. Friedl, "An Illustrated Guide to the Kaminsky DNS Vulnerability," Accessed via: http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

[10] B. Hubert, "DNS Security in the Broadest Sense," Accessed via: http://ds9a.nl/har-presentation-bert-hubert-3.pdf

[11] D. Eastlake, "Domain Name System Security Extensions," RFC 2535 (Proposed Standard), Internet Engineering Task Force,

[12] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Internet Engineering Task Force,

[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Resource Records for the DNS Security Extensions," RFC 4034 (Proposed Standard), Internet Engineering Task Force,

[14] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol Modifications for the DNS Security Extensions," RFC 4035 (Proposed Standard), Internet Engineering Task Force,

[15] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," RFC 5155 (Proposed Standard), Internet Engineering Task Force,

[16] "Diginotar reports security incident," Accessed via: https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx

[17] O. Kolkman, W. Mekking, and R. Gieben, "DNSSEC Operational Practices, Version 2," RFC 6781 (Informational), Internet Engineering Task Force,

[18] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, "Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 656–669, 2011.

[19] J. Jansen, "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC," RFC 5702 (Proposed Standard), Internet Engineering Task Force,

[20] "PKCS #1 v2.2: RSA Cryptography Standard," RSA Laboratories, Accessed via: http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf

[21] T. Pornin, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979 (Informational), Internet Engineering Task Force,

[22] T. Izu, B. Möller, and T. Takagi, "Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks," *Progress in CryptologyINDOCRYPT 2002*, pp. 296–313, 2002. Accessed via: http://www.springerlink.com/index/uc5cvkt1gv9gl6up.pdf

[23] "Measurements of public-key signature systems, indexed by machine," Accessed via: http://bench.cr.yp.to/results-sign.html

[24] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology*, vol. 1109, pp. 104–113, 1996. Accessed via: http://link.springer.com/chapter/10.1007/3-540-68697-5_9

[25] J. S. Coron, "Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems," *Ches*, pp. 292–302, 1999. Accessed via: http://link.springer.com/chapter/10.1007/3-540-48059-5_25

[26] bushing, marcan, segher, and sven, "Console hacking 2010," Accessed via: http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf

[27] R. Chirgwin, "Android bug batters bitcoin wallets," Accessed via: www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets/

[28] C. Burt, "Thousands of French Websites Face DDoS Attacks Since Charlie Hebdo Massacre," Accessed via: http://www.thewhir.com/web-hosting-news/thousands-french-websites-face-ddos-attacks-since-charlie-hebdo-massacre

[29] P. Amsel, "China online gambling bust; Korean site orders DDOS attacks on competitor," Accessed via: http://calvinayre.com/2015/03/03/business/korean-gambling-site-ddos-attack-on-competitor/

[30] T. Sterling and T. Escritt, "Dutch government says DDoS attack took down websites for hours," Accessed via: http://www.reuters.com/article/2015/02/11/us-netherlands-government-websites-idUSKBN0LF0N320150211

[31] S. M. Specht and R. B. Lee, "Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures." in *ISCA PDCS*, 2004, pp. 543–550.

[32] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its Potential for DDoS Attacks - a Comprehensive Measurement Study," in *Proceedings of the Internet Measurement Conference 2014*. Vancouver, BC, Canada: ACM Press, 2014.

[33] G. van den Broek, R. M. van Rijswijk, A. Sperotto, and A. Pras, "DNSSEC Meets Real World: Dealing with Unreachability Caused by Fragmentation," *IEEE Communications Magazine*, vol. 52, no. April, pp. 154–160, 2014. Accessed via: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6828880&sortType%3Ddesc_p_Publication_Year%26queryText%3DDNSSEC

[34] D. Migault, C. Girard, and M. Laurent, "A Performance View on DNSSEC Migration," *Proceedings of the 2010 International Conference on Network and Service Management, CNSM 2010*, pp. 469–474, 2010.

[35] W. C. Wijngaards and B. J. Overeinder, "Securing DNS," *IEEE Security & Privacy*, vol. 7, no. 5, pp. 0036–43, 2009.

[36] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS Amplification Attack Revisited," *Computers and Security*, vol. 39, no. PART B, pp. 475–485, 2013. Accessed via: http://dx.doi.org/10.1016/j.cose.2013.10.001

[37] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, no. February, pp. 23–26, 2014. Accessed via: http://www.internetsociety.org/sites/default/files/01_5.pdf

[38] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous, or: one-domain-to-rule-them-all.org," *2013 IEEE Conference on Communications and Network Security, CNS 2013*, pp. 224–232, 2013.

[39] S. Dickinson, "OpenDNSSEC," Accessed via: https://ripe66.ripe.net/presentations/216-ripe66-dns-wg-opendnssec.pdf

[40] Y. Schaeffer, B. Overeinder, and M. Mekking, "Flexible and Robust Key Rollover in DNSSEC," *Satin*, 2012. Accessed via: http://conferences.npl.co.uk/satin/papers/satin2012-Schaeffer.pdf

[41] R. Chandramouli and S. Rose, "NIST Special Publication 800-81-2,"

[42] A. Herzberg, H. Shulman, and B. Crispo, "Less is More: Cipher-Suite Negotiation for DNSSEC," *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*, no. 3, pp. 346–355, 2014. Accessed via: http://dl.acm.org/citation.cfm?doid=2664243.2664283

[43] A. Herzberg and H. Shulman, "Negotiating DNSSEC Algorithms over Legacy Proxies," pp. 111–126, 2014.

[44] A. Herzberg, "Cipher-Suite Negotiation for DNSSEC: Hop-by-Hop or End-to-End?" 2015.

[45] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS Performance and the Effectiveness of Caching," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 589–603, 2002.

[46] H. Gao, V. Yegneswaran, Y. Chen, P. Porras, S. Ghosh, J. Jiang, and H. Duan, "An Empirical Reexamination of Global DNS Behavior," *SIGCOMM 2013 - Proceedings of the ACM SIGCOMM 2013 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 267–278, 2013. Accessed via: http://dl.acm.org/citation.cfm?doid=2486001.2486018

[47] Y. Koç, A. Jamakovic, and B. Gijsen, "A Global Reference Model of the Domain Name System," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 3-4, pp. 108–117, 2012.

[48] D. Wessels, M. Fomenkov, N. Brownlee, and K. Claffy, "Measurements and Laboratory Simulations of the Upper DNS Hierarchy," *Passive and Active Network Measurement*, pp. 147–157, 2004.

[49] O. Kolkman, "Measuring the Resource Requirements of DNSSEC," pp. 1–24, 2005.

[50] S. Krishnan and F. Monrose, "An Empirical Study of the Performance, Security and Privacy Implications of Domain Name Prefetching," *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 61–72, 2011.

[51] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," *NIST Special Publication*, vol. 800, p. 131A, 2011.

[52] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "Making the Case for Elliptic Curves in DNSSEC," *ACM Computer Communication Review (CCR)*, vol. 45, no. 5, 2015.

[53] K. S. Srikanth, "Theil-Sen Estimator: An alternative to least squares regression," Accessed via: http://talegari.wdfiles.com/local--files/files%3A_nothing/theil_free.pdf

[54] R. O. Gilbert, *Statistical methods for environmental pollution monitoring.* John Wiley & Sons, 1987.

[55] MIT OpenCourseWare, "Kolmogorov-Smirnov Test," Accessed via: http://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-fall-2006/lecture-notes/lecture14.pdf

[56] S. Schwaar, Accessed via: http://www.mathematik.uni-kl.de/~schwaar/Exercises/Tabellen/table_kolmogorov.pdf

[57] "The Domain Name Industry Brief," Verisign, Accessed via: https://www.verisign.com/assets/domain-name-report-june2015.pdf

Appendices

Appendix A

Unbound code

A.1 Store statistics in hash maps

```
/* Increment total number of validations */
nr_validations++;

uint16_t type = sldns_buffer_read_u16(buf); /* type */
sldns_buffer_read_u8(buf); /* algorithm */
sldns_buffer_read_u8(buf); /* number of labels */
sldns_buffer_read_u32(buf); /* original TTL */
sldns_buffer_read_u32(buf); /* expiration */
sldns_buffer_read_u32(buf); /* inception */
sldns_buffer_read_u16(buf); /* key tag */
uint8_t last_char = sldns_buffer_read_u8(buf);
uint8_t label_remaining = last_char;
char sn[255] = "\0";
int dlevel = 0;
int print_dot = 0;
while (last_char != 0){
        if (label_remaining == 0)
        {
                dlevel++;
                sprintf(sn, "%s%c", sn, last_char);
                last_char = sldns_buffer_read_u8(buf);
                label_remaining = last_char;
                print_dot = 1;
        }
        else
        {
                if (print_dot)
                {
                        sprintf(sn, "%s%c", sn, 46);
                        print_dot = 0;
                }
                else
                {
                        sprintf(sn, "%s%c", sn, last_char);
                }
                last_char = sldns_buffer_read_u8(buf);
                label_remaining--;
        }
}
```

```
40 /* Add a trailing '.' to create a FQDN */
41 sprintf(sn, "%s.", sn);
42
43 char *typestr;
44 if (type == 48){
45          typestr = "KSK";
46 }
47 else {
48         typestr = "ZSK";
49 }
50 sldns_buffer_read_u8(buf); /* */
51 last_char = sldns_buffer_read_u8(buf);
52 char name[255] = "\0";
53 int ctr = 1;
54 while (last_char != 0){
55         if (last_char < 32){
56                 last_char = 46;
57         }
58         sprintf(name, "%s%c", name, (unsigned char) last_char);
59         last_char = sldns_buffer_read_u8(buf);
60         ctr ++;
61 }
62 /* Add a trailing '.' to create a FQDN */
63 sprintf(name, "%s.", name);
64
65 /* Store KSK/ZSK keys and their level in the DNS hierarchy */
66 struct ht_cryptoalgo *found_hash_entry = NULL;
67 lookup_key = (struct lookup_key_t*) malloc ( sizeof( struct lookup_key_t) );
68 /* KSK = 0, ZSK = 1 */
69 if (type == 48){
70         lookup_key->algo = 0;
71 }
72 else{
73         lookup_key->algo = 1;
74 }
75 lookup_key->keylen = dlevel;
76
77 HASH_FIND( hh, kskzsk_ht, lookup_key, sizeof(lookup_key), found_hash_entry);
78 if (found_hash_entry){
79         found_hash_entry->occ++;
80 }
81 else {
82         struct ht_cryptoalgo *new_entry = NULL;
83         new_entry = (struct ht_cryptoalgo* ) malloc ( sizeof( struct
              ht_cryptoalgo ) );
84         if (type == 48){
85                 new_entry->algo = 0;
86         }
87         else{
88                 new_entry->algo = 1;
89         }
90         new_entry->keylen = dlevel;
91         new_entry->occ    = 1;
92         HASH_ADD(hh, kskzsk_ht, algo, sizeof(new_entry), new_entry);
93 }
94
95 /* Store signature algorithm and key length */
96 found_hash_entry = NULL;
97 unsigned int rsa_mod = keylen - (unsigned int) *key - 1;
98 lookup_key->algo        = algo;
99
100 /* RSA keys contain the public exponent and should be handled differently*/
```

```
101 if (algo == 1 || algo == 5 || algo == 7 || algo == 8 || algo == 10){
102         lookup_key->keylen = rsa_mod;
103 }
104 /* ECDSA/DSA/GOST keys do not contain the public exponent */
105 else{
106         lookup_key->keylen = keylen;
107 }
108 HASH_FIND( hh, ht, lookup_key, sizeof(lookup_key), found_hash_entry);
109 /* The combination of algorithm and keylengh exists; increment its occurrence
        */
110 if (found_hash_entry){
111         found_hash_entry->occ++;
112 }
113 /* The combination of algorithm and keylength does not exist; create new entry
        */
114 else{
115         struct ht_cryptoalgo *new_entry = NULL;
116         new_entry = (struct ht_cryptoalgo* ) malloc ( sizeof( struct
              ht_cryptoalgo ) );
117         new_entry->algo          = algo;
118         if (algo == 1 || algo == 5 || algo == 7 || algo == 8 || algo == 10)
                  new_entry->keylen = rsa_mod;
119         else
                    new_entry->keylen = keylen;
120         new_entry->occ           = 1;
121         HASH_ADD(hh, ht, algo, sizeof(new_entry), new_entry);
122 }
123 free(lookup_key);
```

A.2 Write data to file

```
1  /* Deinitializes and prints the cryptographic algorithm and key length
       statistics */
2  void deinit_val_secalgo_stat(char* filepath_cryptoalgo, char* filepath_kskzsk)
3  {
4          FILE* fp        = fopen(filepath_cryptoalgo, "a");
5          if ( fp != NULL)
6          {
7                  struct ht_cryptoalgo *tmp               = NULL;
8                  struct ht_cryptoalgo *iter_entry        = NULL;
9
10                 /* Only print if the hash table is not empty */
11                 if (ht != NULL)
12                 {
13                         HASH_SORT(ht, sort_on_occ_descending);
14                         HASH_ITER(hh, ht, iter_entry, tmp)
15                         {
16                                 fprintf(fp, "%d\t%d\t%d\n", iter_entry->algo,
                                       iter_entry->keylen, iter_entry->occ);
17                                 HASH_DEL( ht, iter_entry);
18                                 free(iter_entry);
19                         }
20                         fprintf(fp, "total\t%d\n", nr_validations);
21                         ht = NULL;
22                 }
23                 else
24                 {
25                         fprintf(fp, "total\t0\n");
26                 }
27                 fprintf(fp, "\n\n"); /* Empty lines for Gnuplot processing */
28                 fflush(fp);
29                 fclose(fp);
30         }
31         fp      = fopen(filepath_kskzsk, "a");
32         if ( fp != NULL)
33         {
34                 struct ht_cryptoalgo *tmp               = NULL;
35                 struct ht_cryptoalgo *iter_entry        = NULL;
36
37                 /* Only print if the hash table is not empty */
38                 if (kskzsk_ht != NULL)
39                 {
40                         HASH_SORT(kskzsk_ht, sort_on_occ_descending);
41                         HASH_ITER(hh, kskzsk_ht, iter_entry, tmp)
42                         {
43                                 fprintf(fp, "%d\t%d\t%d\n", iter_entry->algo,
                                       iter_entry->keylen, iter_entry->occ);
44                                 HASH_DEL( kskzsk_ht, iter_entry);
45                                 free(iter_entry);
46                         }
47                         fprintf(fp, "total\t%d\n", nr_validations);
48                         kskzsk_ht = NULL;
49                 }
50                 else {
51                         fprintf(fp, "total\t0\n");
52                 }
53                 fprintf(fp, "\n\n"); /* Empty lines for Gnuplot processing */
54                 fflush(fp); fclose(fp);
55         }
56 }
```

Appendix B

Results of regression

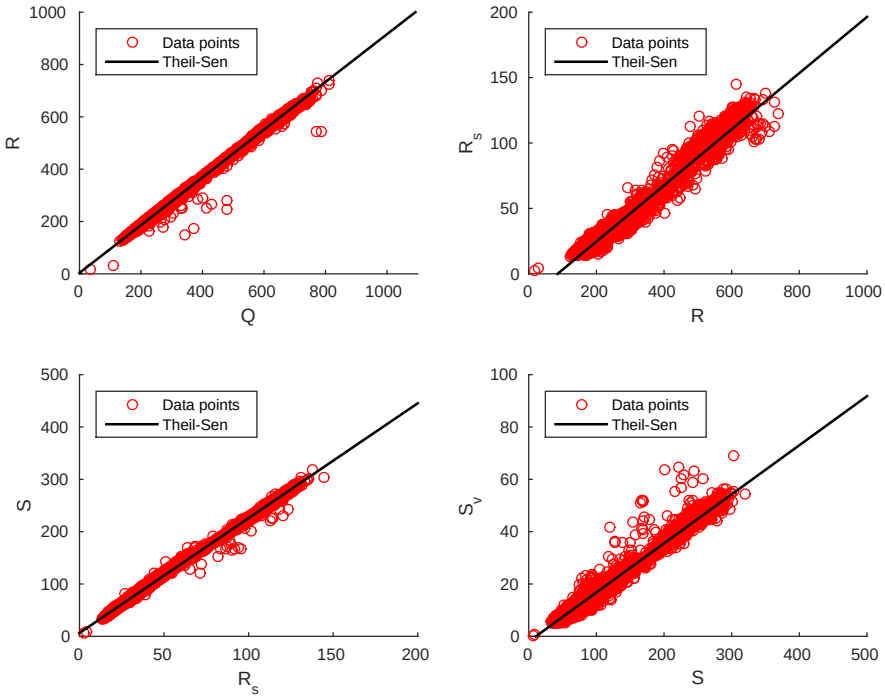


Figure B.1: AMS

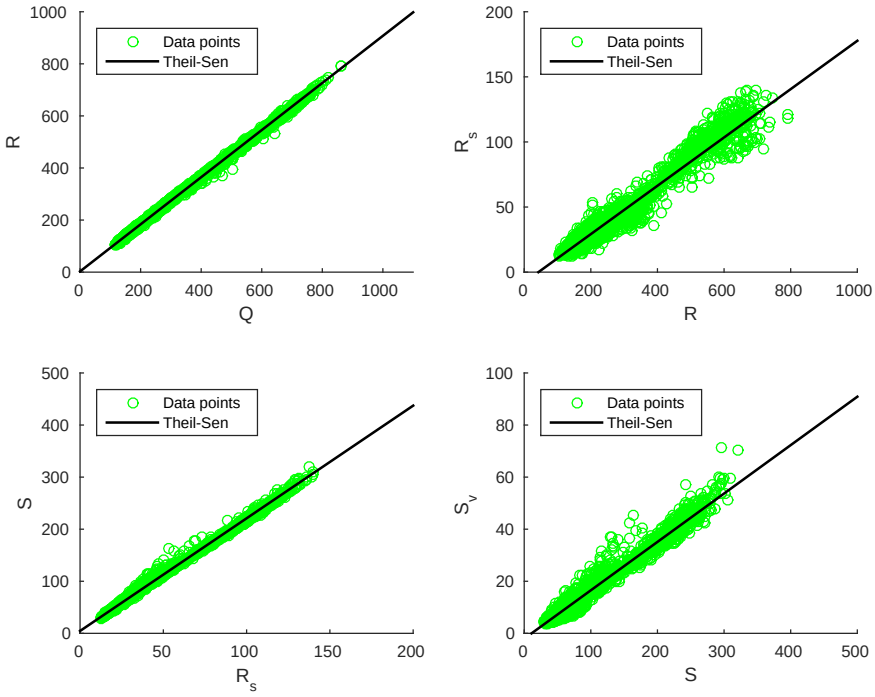


Figure B.2: TIL

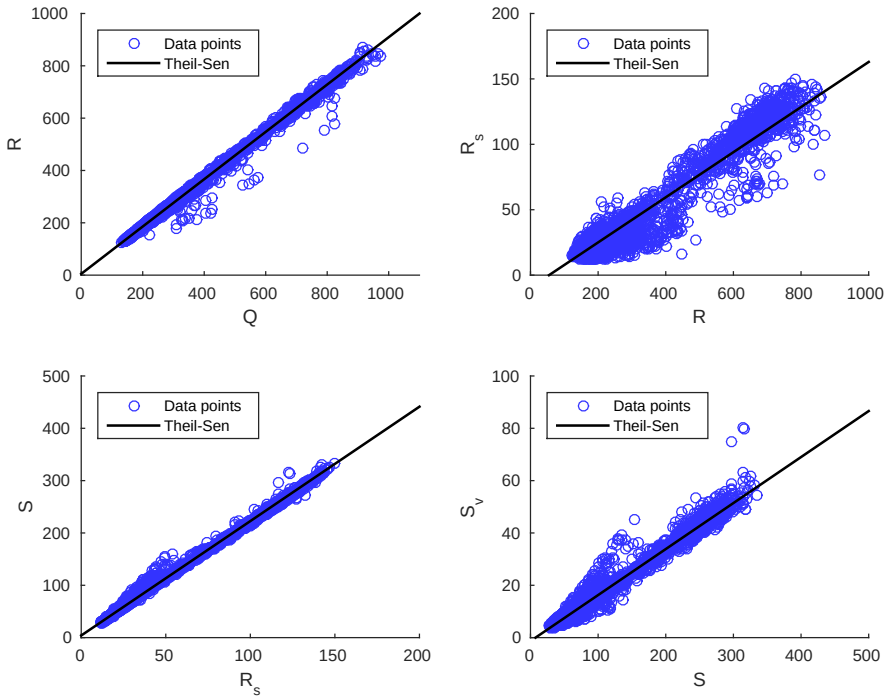


Figure B.3: UTR

Appendix C

CPU load code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <limits.h>
#include "openssl/conf.h"
#include "openssl/engine.h"
#include "openssl/err.h"
#include "openssl/pem.h"
#include "openssl/evp.h"
#include <sys/time.h>
#include <sys/resource.h>

#include "ed25519/src/ed25519.h"
#include "ed25519/src/ge.h"
#include "ed25519/src/sc.h"

//#define ED25519_DLL
//#define DEBUG

typedef enum {
   PROC,
   SYS
} usage_type;

static const int USEC_PER_SEC = 1000000;
static const int MAX_WARNINGS = 100;
static const int ITERATIONS     = 50;
static const EVP_MD * type;
unsigned int hash_len, sig_len;

unsigned char* sha(char * input)
{
    EVP_MD_CTX c;
    unsigned char *hash;
    unsigned int len;

    hash = malloc(EVP_MAX_MD_SIZE);

    EVP_MD_CTX_init(&c);
    EVP_DigestInit_ex(&c, type, NULL);
    EVP_DigestUpdate(&c, input, strlen(input));
```

```
43      EVP_DigestFinal_ex(&c, hash, &len);
44      EVP_MD_CTX_cleanup(&c);
45
46      hash_len = len;
47      return hash;
48  }
49
50  unsigned char* sign(EVP_PKEY * key, unsigned char * hash)
51  {
52      EVP_MD_CTX c;
53      unsigned char *sig;
54      unsigned int len;
55
56      EVP_MD_CTX_init(&c);
57      sig = malloc(EVP_PKEY_size(key));
58
59      EVP_SignInit(&c, type);
60      EVP_SignUpdate(&c, hash, strlen((char *)hash));
61      EVP_SignFinal(&c, sig, &len, key);
62
63      EVP_MD_CTX_cleanup(&c);
64
65      sig_len = len;
66      return sig;
67  }
68
69  int verify(EVP_PKEY *key, unsigned char *sig, unsigned int sig_len, unsigned
        char *hash, unsigned int hash_len)
70  {
71      EVP_MD_CTX c;
72      int ret;
73
74      EVP_MD_CTX_init(&c);
75      EVP_VerifyInit(&c, type);
76      EVP_VerifyUpdate(&c, hash, hash_len);
77      ret = EVP_VerifyFinal(&c, sig, sig_len, key);
78
79      return ret;
80  }
81
82  int arguments_check(int argc, char *argv[])
83  {
84     if (argc != 4)
85     {
86        printf("Wrong arguments: <algorithm, duration, maximum validations per
              second>\n");
87        return -1;
88     }
89     else if (strcmp(argv[1], "rsa1024") != 0 && strcmp(argv[1], "rsa2048") != 0
           && strcmp(argv[1], "ecdsap256") != 0 && strcmp(argv[1], "ecdsap384") !=
           0 && strcmp(argv[1], "ed25519") != 0)
90     {
91        printf("Given algorithm '%s' is not supported\n", argv[1]);
92        return -1;
93     }
94     return 0;
95  }
96
97  /* Returns the secret key file for the given algorithm */
98  FILE* get_sfile(char* algo)
99  {
100    FILE* res;
```

```
101     char *filestr = (char*)malloc(256);
102     sprintf(filestr, "/home/kaspar/cpuspeed/keys/%ssec.pem", algo);
103     res = fopen(filestr, "r");
104     free(filestr);
105     return res;
106 }
107 
108 /* Returns the public key file for the given algorithm */
109 FILE* get_pfile(char* algo)
110 {
111     FILE* res;
112     char *filestr = (char*)malloc(256);
113     sprintf(filestr, "/home/kaspar/cpuspeed/keys/%spub.pem", algo);
114     res = fopen(filestr, "r");
115     free(filestr);
116     return res;
117 }
118 
119 char* get_plaintxt(void)
120 {
121     FILE *ptfd;
122     char *res;
123     long length;
124     ptfd = fopen("/home/kaspar/cpuspeed/plaintxt", "r");
125     if (ptfd)
126     {
127         fseek (ptfd, 0, SEEK_END);
128         length = ftell (ptfd);
129         fseek (ptfd, 0, SEEK_SET);
130         res = malloc (length);
131         if (res)
132         {
133         fread (res, 1, length, ptfd);
134         }
135         fclose (ptfd);
136     }
137     return res;
138 }
139 
140 int get_timestamp(void)
141 {
142     int res;
143     struct timeval tv;
144     gettimeofday(&tv,NULL);
145     res = tv.tv_sec*USEC_PER_SEC+ tv.tv_usec;
146     return res;
147 }
148 
149 /* Read the CPU usage for either the current process or for the system */
150 int get_usage(usage_type type)
151 {
152 
153     FILE* fp;
154     if (type == SYS)
155     {
156         double long out1,out2,out3,out4;
157         fp = fopen("/proc/stat","r");
158         fscanf(fp,"%*s %Lf %Lf %Lf %Lf", &out1, &out2, &out3, &out4);
159         fclose(fp);
160         return out1+out2+out3+out4;
161     }
162     else if (type == PROC)
```

```
163     {
164         double long out1, out2;
165         int pid = getpid();
166         char *filestr = (char *)malloc(256);
167         sprintf(filestr, "/proc/%d/stat", pid);
168         fp = fopen(filestr,"r");
169         fscanf(fp,"%*f %*s %*s %*f %*f %*f %*f %*f %*f %*f %*f %*f %*f %*f %Lf %Lf ",
                &out1, &out2);
170         fclose(fp);
171         return out1+out2;
172     }
173     else
174     {
175         printf("Unsupported type");
176         exit(-1);
177     }
178     return 0;
179 }
180
181 int main(int argc, char *argv[])
182 {
183     /* Check arguments */
184     if (arguments_check(argc, argv) != 0)
185     {
186         exit(-1);
187     }
188
189     /* In case of ECDSA and RSA, use OpenSSL */
190     if (strcmp(argv[1], "rsa1024") == 0 || strcmp(argv[1], "rsa2048") == 0 ||
            strcmp(argv[1], "ecdsap256") == 0 || strcmp(argv[1], "ecdsap384") == 0)
191     {
192         EVP_PKEY *sk, *pk;
193         FILE *sfd, *pfd;
194         unsigned char *hash, *sig;
195         unsigned int runtime, uval_before, uval_after, usys_before, usys_after, i
                , err, success, fail, valctr;
196         int t_before, t_passed;
197         double target_vals_per_sec, multiplier;
198         char *pt;
199
200         /* Read the public- and secret key from file */
201         sfd = get_sfile(argv[1]);
202         pfd = get_pfile(argv[1]);
203         sk = PEM_read_PrivateKey(sfd, NULL, NULL, NULL);
204         pk = PEM_read_PUBKEY(pfd, NULL, NULL, NULL);
205         fclose(sfd);
206         fclose(pfd);
207
208         /* Read the plain text from file;      */
209         pt = get_plaintxt();
210
211         /* Obtain the correct EVP_MD structure for the hash function*/
212         OpenSSL_add_all_digests();
213         if (strcmp(argv[1], "rsa1024") == 0 || strcmp(argv[1], "rsa2048") == 0)
214         {
215             type = EVP_get_digestbyname("SHA1");
216         }
217         else if (strcmp(argv[1], "ecdsap256") == 0)
218         {
219             type = EVP_get_digestbyname("SHA1");
220         }
221         else if (strcmp(argv[1], "ecdsap384") == 0)
```

```
222         {
223            type = EVP_get_digestbyname("SHA384");
224         }
225         else
226         {
227            printf("Algorithm not supported\n");
228            exit(-1);
229         }
230
231         if (type == NULL)
232            {
233               printf("This version of OpenSSL (%lx) does not support the
                       requested algorithm", OPENSSL_VERSION_NUMBER);
234               exit(01);
235            }
236
237         runtime     = atoi(argv[2])*USEC_PER_SEC;
238
239         /* Create the hash value and sign it */
240         hash        = sha(pt);
241         sig         = sign(sk, hash);
242
243
244         for(i = 1; i <= ITERATIONS; i++)
245         {
246            valctr        = 0;
247            err         = 0;
248            success     = 0;
249            fail        = 0;
250            multiplier     = (double)i/(double)ITERATIONS;
251            target_vals_per_sec = (double)(atoi(argv[3])*multiplier);
252            t_before = get_timestamp();
253            uval_before = get_usage(PROC);
254            usys_before = get_usage(SYS);
255            t_passed = 0;
256
257            while(valctr < target_vals_per_sec*atoi(argv[2]) && t_passed < runtime
                   )
258            {
259
260               /* Verify the signature */
261               switch( verify(pk, sig, sig_len, hash, hash_len) )
262               {
263                  case 0:
264                     fail++;
265                     break;
266                  case 1:
267                     success++;
268                     break;
269                  default:
270                     err++;
271                     printf("An error occurred: %lu\n", ERR_get_error());
272                     break;
273               }
274               t_passed = get_timestamp()-t_before;
275               valctr++;
276            }
277
278            t_passed = get_timestamp()-t_before;
279            if ((signed int) runtime-t_passed > 0)
280            {
281               usleep(runtime-t_passed);
```

```
282             }
283
284             uval_after = get_usage(PROC);
285             usys_after = get_usage(SYS);
286             double cpuload = (double)(uval_after - uval_before)/(usys_after -
                    usys_before)*100;
287             printf(   "%.1f\t\t%.1f\t\t%.4f\n", target_vals_per_sec, (double)(
                    valctr/(double)atoi(argv[2])), cpuload);
288         }
289         EVP_cleanup();
290         free(hash);
291         free(sig);
292     }
293     /* In case of ED25519, use custom implementation */
294     else if (strcmp(argv[1], "ed25519") == 0)
295     {
296        unsigned char public_key[32], private_key[64], seed[32], signature[64];
297        unsigned int runtime, uval_before, uval_after, usys_before, usys_after,
               i, success, fail, valctr;
298         int t_before, t_passed;
299         double target_vals_per_sec, multiplier;
300
301         runtime    = atoi(argv[2])*USEC_PER_SEC;
302
303         const unsigned char pt[] = "This is a DNS RRset!";
304         const int pt_len = strlen((char *) pt);
305
306         ed25519_create_seed(seed);
307         ed25519_create_keypair(public_key, private_key, seed);
308
309         /* create signature on the message with the keypair */
310         ed25519_sign(signature, pt, pt_len, public_key, private_key);
311
312         for(i = 1; i <= ITERATIONS; i++)
313         {
314            valctr       = 0;
315            success    = 0;
316            fail      = 0;
317            multiplier    = (double)i/(double)ITERATIONS;
318            target_vals_per_sec = (double)(atoi(argv[3])*multiplier);
319            t_before = get_timestamp();
320            uval_before = get_usage(PROC);
321            usys_before = get_usage(SYS);
322            t_passed = 0;
323
324            while(valctr < target_vals_per_sec*atoi(argv[2]) && t_passed < runtime
                   )
325            {
326               /* Verify the signature */
327               if (ed25519_verify(signature, pt, pt_len, public_key))
328               {
329                  success++;
330               }
331               else
332               {
333                  fail++;
334               }
335               t_passed = get_timestamp()-t_before;
336               valctr++;
337            }
338
339            t_passed = get_timestamp()-t_before;
```

```
340             if ((signed int) runtime-t_passed > 0)
341             {
342                 usleep(runtime-t_passed);
343             }
344 
345             uval_after = get_usage(PROC);
346             usys_after = get_usage(SYS);
347         #ifdef DEBUG
348             printf("uval_diff: %d\n", uval_after - uval_before);
349             printf("usys_diff: %d\n", usys_after - usys_before);
350         #endif
351             double cpuload = (double)(uval_after - uval_before)/(usys_after -
                    usys_before)*100;
352             printf(    "%.1f\t\t%.1f\t\t%.4f\n", target_vals_per_sec, (double)(
                    valctr/(double)atoi(argv[2])), cpuload);
353         }
354     }
355     return 0;
356 }
```