Is there a case to prefer Ed25519 over ECDSA P-256 for DNSSEC?

J.J. Yu
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
j.yu@student.utwente.nl

ABSTRACT

DNS security Extensions (DNSSEC) adds cryptographic signatures to DNS. The most popular cryptographic signature algorithm in DNSSEC is RSA as of June 2017. Using RSA could cause problems due to the relatively large key sizes that are required, such as packet fragmentation and DNSSEC servers being turned into an amplification vector for distributed denial-of-service (DDoS) attacks, because of the large amount of information that is returned.

Cryptographic signature schemes based on Elliptic Curve Cryptography (ECC) gained popularity in DNSSEC due to the smaller key sizes that are required to get a 128-bit security level compared to RSA. ECC's smaller keys and signatures promise to resolve the fragmentation and amplification issues mentioned before.

As of June 2017, the most popular elliptic curve in DNSSEC is the NIST curve P-256. A newer elliptic curve algorithm, Ed25519, which uses a so-called Edwards curve has been standardized for use in DNSSEC in February 2017, citing security problems with the currently used elliptic curves as a motivation. Ed25519 can be seen as an alternative for P-256, because both have small key sizes and are at the ~128-bit security level.

While Ed25519 has promising properties for DNSSEC such as speed and security, it is unclear whether it should be preferred over P-256. Therefore, in this paper we study the question: Is it worth switching from using P-256 signatures to Ed25519 signatures in DNSSEC?

In order to evaluate this, the security problems and the performance on a variety of hardware architectures that reflect common computing platforms, such as servers and small home routers were studied.

Security concerns related to insecure implementations are already being addressed in modern implementations of P-256. A remaining security concern could be the trustworthiness of P-256. The performance of Ed25519 and P-256 were similar when comparing the fast assembly language implementation of P-256 and the reference implementation of Ed25519 that were available in OpenSSL as of June 2017. For an optimal performance of Ed25519 fast assembly language implementations of Ed25519 are preferable however, which could then make the speed and trust aspects worth it to switch to Ed25519 if it becomes widely available.

Keywords

DNSSEC, ECDSA, EdDSA

1. INTRODUCTION

DNS translates human readable domain names into machine readable information such as IP-addresses. DNSSEC adds signatures to DNS to provide authenticity and integrity.

Van Rijswijk-Deij et al. found that DNSSEC could cause large DDoS amplification factors [1] and fragmentation [2] because of the large RSA key sizes that were required to provide a sufficient amount of security. Van Rijswijk-Deij et al. proposed to use Elliptic Curve Cryptography (ECC) instead, as ECC would be able to provide equivalent or better cryptographic security with smaller key sizes, which promise to resolve the fragmentation and amplification issues mentioned before. [3].

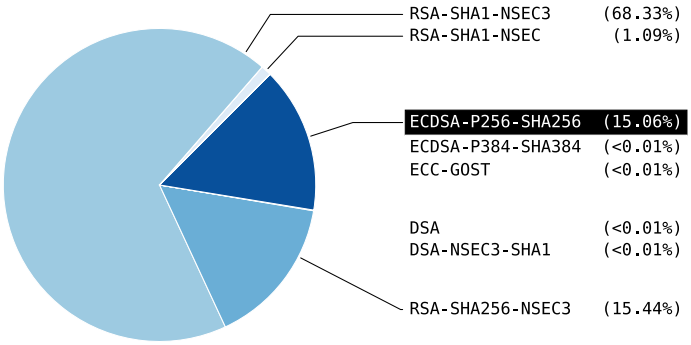

Figure 1: DNSSEC stats for .com for 2017-06-15 [4].

There are different types of curves that can be used for Elliptic Curve Cryptography. This paper will only focus on two of them: The NIST P-256 curve and the Ed25519 curve. The NIST P-256 curve is a so-called Weierstrass curve. Signature generation and signature validation using this curve are done with the Elliptic Curve Digital Signature Algorithm (ECDSA). The Ed25519 curve (formally called the "edwards25519 curve" [5]) is a so-called twisted Edwards curve. Signature generation and signature validation using this curve are done with the Edwards-curve Digital Signature Algorithm (EdDSA), which is a modified version of ECDSA.

As shown in figure 1, P-256 is the most popular elliptic curve in DNSSEC as of June 2017. Ed25519 was standardized for use in DNSSEC through RFC8080 [6] in February 2017, citing security problems with the currently used elliptic curves such as P-256 as a motivation [7] [8].

What makes Ed25519 comparable to P-256 is that they both have approximately the same security level and both have small key sizes. For P-256 the public key size is 64 bytes [9] and for Ed25519 the public key size is 32 bytes [6].

Using ECC also requires extra load on the resolver in order to validate signatures. In 2016, Van Rijswijk-Deij et al. et al concluded that resolvers are able to handle the load that was required to validate ECC signatures on x86/x64, but also that Ed25519 was faster than ECDSA using curve P-256 for verifying signatures [10] and would therefore be able to reduce the load even more.

In this work the benchmarks are done on other architectures as well. As DNSSEC software can be run on other architectures as well, such as an ARM based Raspberry Pi or a MIPS based OpenWrt home router. Which could then be used to replace a third-party DNSSEC-validating resolver that could block domains or monitor all the domains that are visited.

Since signature generation speed is also important for DNSSEC servers that generate many signatures per second the performance of signature generation is also included.

So while Ed25519 has promising properties for DNSSEC such as security and performance, it is unclear whether it should be preferred over P-256.

The main research question studied in this paper therefore is: Is it worth switching from P-256 signatures to Ed25519 signatures in DNSSEC?

The sub questions are:

- Does Ed25519 offer better security than ECDSA P-256 and and are the differences relevant enough to prefer Ed25519 signatures over P-256 signatures in DNSSEC?
- How does the performance of ECDSA P-256 compare to Ed25519 on different hardware architectures?

2. Related work

2.1 Security

Some work that is related to the security of ECDSA P-256 and Ed25519 has been done before. Nystrom [11] noted during a review of an early RFC8080 draft that there were no references or proof that Ed25519 would offer "improved security properties and implementation characteristics compared to RSA and ECDSA algorithms", which resulted this statement to be removed from RFC8080. There could be reasons to believe that Ed25519 would be more secure than ECDSA P-256 however. E.g. when checking Lange and Bernstein's security checklist for elliptic curves in general called SafeCurves [12], it can be noticed that P-256 is considered to be unsafe, while Curve25519 (which is related to Ed25519) is considered to be safe. The relevance and impact of it to DNSSEC was not evaluated however. An ECDSA P-256 specific attack has been described as well. Brumley et al. [13] found that ECDSA P-256 in the latest version of OpenSSL 1.0.1 (which is OpenSSL 1.0.1u) is vulnerable to cache-timing attacks, allowing them to recover the private key for TLS and SSH. Which could be relevant for DNSSEC, as DNSSEC software can rely OpenSSL. Whereas implementations of Ed25519 could be protected against cache-timing attacks [14]. In this work the significance of these security problems and possible mitigations are evaluated for DNSSEC.

2.2 Performance

Some related benchmarks for Ed25519 and ECDSA P-256 implementations have been done already. Existing benchmarks show that there are indeed performance differences between different implementations. For example: bench.cr.yp.to [15] lists benchmarking results for Ed25519 in SUPERCOP, which show that the optimized amd64-64-24k version of Ed25519 is on average approximately 2.5 as fast the ref10 reference implementation. There are also speed improvements for ECDSA P-256. The authors of the optimized NIZT256 implementation of ECDSA P-256 [16] claim that it can be 3 times as fast as the non-optimized version, but did not compare it to Ed25519. Rijswijk et al. [10] benchmarked ECDSA P-256 and Ed25519 on an Intel processor and compared them, showing that Ed25519 in Ed25519-donna is approximately 1.4 times as fast as ECDSA P-256 in OpenSSL 1.0.2e on an Intel processor. While this work focuses on comparing several implementations of Ed25519 and ECDSA P-256 on x64, ARM and MIPS to reflect that DNSSEC software can be used on other architectures with other implementations of Ed25519 and ECDSA P-256 as well.

3. Background information

3.1 Signatures in DNSSEC

DNSSEC is meant to improve the security of DNS by adding cryptographic signature to DNS, which add authenticity and integrity to DNS.As described in RFC6781 [17] the term 'key' is used loosely in DNSSEC, as the reader is expected to familiar with public key cryptography and understands that the private part of the key pair is used for signing and the public part of the key pair is published in the DNSKEY resource record.

In a DNSSEC signed DNS zone there usually uses two types of keys: The Zone Signing Key (ZSK) and the Key Signing Key (KSK). The KSK is published in the DNSKEY resource record to serve as a trust anchor and signs the ZSK. The ZSK signs the other data in the zone. Instead of using two keys it is also possible to use one Combined Signing Key (CSK) that is published in the DNSKEY resource record and signs all the data in the zone.

The popularity of the cryptographic algorithms for .COM are shown in figure 1. As can be seen from figure 1, P-256 is the most popular elliptic curve in DNSSEC as of June 2017. While RSA is the most popular algorithm for signatures. With SHA1 still being the most popular hash algorithm, despite that for SHA1 it has been shown that it is feasible to produce a hash collision in which 2 messages produce the same SHA1 hash by Stevens et al [18]. Practical hash collisions are undesired, as it could allow an attacker to generate a new message for which an earlier generated signature would be valid as well.

3.2 Elliptic Curve Cryptography

Elliptic Curve Cryptography is a form of public key cryptography from which the security relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP). The Elliptic Curve Discrete Logarithm Problem is that when given a base point P and another point on the curve Q, which represents the public key, it is hard to find the multiplicand d, which is the secret key, such that Q =dP.

The approximately 128-bit security level of Ed25519 and P-256 is referring to the approximately 2^{128} mathematical operations that are required to solve the Elliptic Curve Discrete Logarithm problem using the state-of-the-art Pollard's rho method [12]. Doing approximately 2^{128} operations in a reasonable amount of time is considered to be infeasible however.

That the Elliptic Curve Discrete Logarithm Problem is hard to solve is important for ECC-based signature systems to prevent attackers from recovering the private key by solving the Elliptic Curve Discrete Logarithm Problem. Recovering the private key would allow an attacker to forge signatures, which would break the security that a signature system is intended to offer.

Signature algorithms are required to generate and verify DNSSEC signatures. ECDSA (Elliptic Curve Digital Signature Algorithm) [19] and EdDSA (Edwards-curve Digital Signature Algorithm) use elliptic curves for 'elliptic curve point multiplications' (also known as 'scalar multiplications'). Which is an operation that is required for key pair generation, signature generation and signature verification.

The two signature algorithms mentioned above can be implemented on different curves. In this paper, we look at ECDSA using the NIST P-256 curve. This curve is a so-called Weierstrass curve, which is the classical form for elliptic curves.

We also look at EdDSA. Ed25519 is an instance of EdDSA that is instantiated with the 'edwards25519' parameters [5]. Ed25519 signatures use a twisted Edwards curve, because of the speed improvements when using twisted Edwards curves [14].

This twisted Edwards curve can be transformed into Curve25519 by using a substitution and transformed back by using another substitution, which makes it birationally equivalent to Curve25519 [20]. This makes it just as hard to solve the Elliptic Curve Discrete Logarithm Problem on this curve as on Curve25519. [14]

Curve25519 is a so-called Montgomery curve. The Elliptic Curve Diffie–Hellman (ECDH) key exchange function that is using Curve25519 is referred to as X25519 [20]. Therefore, X25519 should not be confused with Ed25519.

4. Methodology

In this section the methodology to answer the research questions about security and performance are described.

4.1 Security

The security of elliptic curve cryptography is evaluated based on what has been described in the literature. After which potential security issues, their relevance to DNSSEC signatures and possible solutions are evaluated.

4.2 Performance

In order to generate and verify DNSSEC signatures DNSSEC software is used. Benchmarks are done to get an idea what the upper bound is of how many signatures and validations per second can be done to see if and how much speed improvements can be gained when switching from ECDSA P-256 to Ed25519 when assuming highly optimized DNSSEC software implementations in which the maximum speed of

signing and validating primarily depends on the maximum speed of the signature algorithm itself and not on other kind of delays.

Benchmarks are done on different hardware architectures (x64, ARM and MIPS) to reflect that DNSSEC software cannot only be used on x64 based devices, but also on ARM and MIPS based devices, such as routers.

Different implementations of ECDSA P-256 and Ed25519 are benchmarked to reflect the popularity of the implementation and performance differences due to optimizations that are present in the implementations. E.g. benchmarks for Ed25519 and an optimized implementation of P-256 that is called NISTZ256 (also known as ecp_nistz256) [21] in OpenSSL are included, because there are popular DNSSEC software implementations (such as BIND) that use OpenSSL.

For the benchmarking consecutives signature generations and signature verifications are done for 10 seconds after which the average amount of signatures/second and verifications/second are calculated first. This process is then repeated 100 times after which the mean and standard deviation over these 100 runs are calculated.

5. Security

In this section the results of the analysis of the security differences between ECDSA P-256 and Ed25519 are examined.

This first parts focuses on ECDSA specific weaknesses, the second part focuses on the SafeCurve criteria and the third part focuses on a practical example in which a flaw in cryptographic implementations could lead to key compromise.

5.1 ECDSA weaknesses

A weakness of ECDSA is that it does not only requires the secret key to be secret. In the signing process, ECDSA also uses a nonce, called 'k'. This nonce needs to remain a secret as well. As when this nonce is known the private key can be derived and signatures can be forged.

Information about the secret nonce can be leaked because of unreliable random number generators and timing attacks. These issues are discussed in the next subsections of the paper.

5.1.1 Unreliable random number generators

When the same nonce is used for two different messages, the nonce can be derived [22]. After which the secret key can be derived. In order to avoid this, ECDSA has traditionally been using randomly generated nonces, but when the random number generator that is used to generate the random nonce would generate the same number twice, you could get two different messages with the same nonce and the same problem occurs. If you can guess the nonce value correctly, you could derive the private key as well.

In order to avoid this problem, several solutions have been proposed based on the idea of that you can get a unique identifier for each message that is also hard to predict, by hashing the private key concatenated with the message. In which including the message as an input is intended to make the output different for each message and including the private key as input is intended to make the output hard to predict when you don't know the private key.

Variations of this idea have been used in deterministic ECDSA [23], EdDSA (including Ed25519) [17] and an OpenSSL patch in 2013 by Langley [24], but they all have to same goal, which is to avoid that you get predictable nonces or even the same nonce for two different messages, when you have a predictable random number generator.

5.1.2 Timing attacks

When a few bits of the nonce are known for several nonces, then the private key could be recovered through lattice techniques [25] [26]. Information about a few bits of the nonces could not only be obtained when a predictable random number generator is used to generate the nonces, but also by measuring the time an operation takes if the execution time is not constant [27]. Such attacks are called timing attacks. Timing attacks can be prevented by using constant-time implementations. A constant-time implementation would run operations that depend on secret data in a constant amount of time and avoids conditional branches that depend on secret data, memory access patterns that depend on secret data or operations that depend on secret data that have a variable execution time. [28] This will be described in more detail in the next subsections.

5.1.2.1 Conditional branches

A conditional branch (e.g. if-then-else statements, for and while loops) directs a program to to a new memory address if a certain condition is true. Conditional branches that could give away secret information must be avoided.

An example of such a conditional branch is described by Brumley and Tuveri [27] in which there was an optimization that allowed the leading bits of the nonce that were zero to be skipped, which caused the scalar multiplications that are required for ECDSA signing operations to be faster for small nonce values than for large nonce values. In which case it can be assumed that the fastest responses for signing operations use nonces with small values which have a certain amount of leading bits that are zero. After many signing operations several of these signatures with fast response times can be collected. From which it is known that nonces some leading bits that are zero were used, because they resulted into fast scalar multiplications and therefore fast responses. Which allows the private key to be recovered through lattice techniques.

Note that for generating the signature not only a scalar multiplication with the secret nonce has to be calculated, but also the inverse of the secret nonce (k^{-1}). When either of them has conditional branches, memory access patterns that depend on the secret nonce or operations that are not constant-time, information could be leaked about the secret nonce. [13] [29]

5.1.2.2 Cache timing attacks

Processor caches are intended to provide faster access to information that was accessed before by storing information in the cache. This could allow a cache-timing attack in which a spy process (that could be in a virtual machine) on the same physical computer as the victim checks if some information was accessed before or not by the victim by measuring the time it takes to access the information. When specific information is accessed or not depending on the value of the secret nonce, you have a memory access pattern that depends on the secret nonce and information could be obtained by the spy process about some bits of the nonce by checking if that information is in the cache or not. When several bits of the nonces for several nonces are obtained the private key can be recovered through lattice techniques.

Solutions have been proposed to always load in more information in the cache than is directly needed and use arithmetic to select the right information, so an attacker can't tell if the information was directly used or not by what is stored in the cache.

Constant-time solutions to prevent such timing attacks have been available in several Ed25519 [14] implementations and the NISTZ256 [21] implementation of P-256 in OpenSSL 1.0.2, 1.1.0 and higher that is based on the work of Gueron and Krasnov [29].

5.2 SafeCurves

In 2013 Bernstein and Lange started a website called SafeCurves that lists criteria for a structured analysis of the security of different elliptic curves. In this section, it is evaluated how ECDSA P-256 and Ed25519 compare, based on these criteria. The focus is on how any of the described security problems are, or are not relevant in the context of DNSSEC.

When ECDSA P-256 and Ed25519 are evaluated according to the SafeCurves criteria, it can be observed that Curve25519 passes all the criteria. This result is applicable to Ed25519 as well, because the used curve is equivalent to Curve25519.

P-256, however, does not pass all criteria. In particular, the following four criteria:

1. Rigidity: Which is about trust in the curve parameters.
2. Completeness: Which is about the completeness of the addition formulas
3. Indistinguishability: Which is about a way to to encode a point on the curve as a uniform random string.

4. Ladders: Which is referring to a way in which scalar multiplications on an elliptic curve could be calculated.

So these four criteria are evaluated for their relevance.

5.2.1 Rigidity

Some trust the curve parameters of Curve25519 more than the curve parameters of NIST P-256, because there's a better explanation of how the exact values that are used are chosen [12].

One reason why people are concerned with parameters could be because of DUAL_EC_DRBG [30]. DUAL_EC_DRBG was a random-looking number generator that was using elliptic curve cryptography for which the parameters can be been chosen in such a way that would allow an attacker with the correct private key predict the subsequent outputs of the random-looking number generator correctly after observing an output [30].

The argumentation for the Curve25519 parameter choices include small values and efficiency at a "comfortable security level" [12].

Coefficients of NIST P-256 are generated by hashing the seed "c49d3608 86e70493 6a6678e1 139d26b7 819f7e90" without an explanation on why this seed was chosen. Which led to speculation on if P-256 curve would be vulnerable to a secret attack that applies to a for example one in a billion curves or not [12].

If this is really an issue, this would imply that there is an extremely large group of weak curves, which would be unlikely to remain undiscovered by other researchers as well [31]. The relevance of rigidity to DNSSEC would be that some users might feel more comfortable with curves they feel that are more trustworthy and could prefer them for this reason.

5.2.2 Completeness

Elliptic curve cryptography relies on elliptic curve point multiplications. These elliptic curve point multiplications can then be expressed as elliptic curve point additions. An elliptic curve point addition is an operation to add two points on an elliptic curve. In order to calculate the results of these additions so-called addition formulas are used.

Completeness is referring to complete addition formulas. A complete addition formula means that the same formula can be used to add any two points on the curve without exceptions for if the input points are equal, opposite or at the neutral element.

There is a complete addition formula for Edwards curves [32]. Which could make it easier for ECC algorithm implementers to make a correct implementation. When not all the exceptional cases are implemented correctly, it can lead to incorrect results (such as division by zero errors).

There are also complete addition formulas for Weierstrass curves such as P-256 [33] [34], but these formulas incur a performance penalty. When no complete addition formulas are used, ECC algorithm implementers would have to check carefully if every exceptional case is implemented correctly and correct it if that would not be the case.

On the Safecurves page on completeness, Izo and Takagi's "exceptional procedure attack" [35] is described. This "exceptional procedure attack" could cause leakage of information about the private key when the exceptional cases are not implemented correctly. As Izo and Takagi already have concluded, this attack is "not relevant to ECDSA because the base point of ECDSA is usually fixed as the system parameter". The reason the base point is considered to be fixed for standardized curves is that the basepoint that must be used is the one that is defined in the standard documents. Therefore, this does not apply to DNSSEC.

5.2.3 Indistinguishability

Elliptic curve points that are indistinguishable from a uniform random string is not a property that is needed for signatures in DNSSEC and therefore irrelevant. It is relevant in, for example, key exchange protocols that want the transmitted content, including the key exchange itself, to be indistinguishable from random data for eavesdroppers to bypass censorship.

5.2.4 Ladders

Lange and Bernstein recommend ladders for calculating constant time scalar multiplications. Which is a recommendation for people who implement ECC algorithms. For end-users the important part here is constant time operations are used for signing operations. Since non-constant time operations could give an attacker information (hints) that could be used to recover the private key that is used, as discussed in section 5.1.2.

Examples of DNSSEC implementations that support online/on-demand signing are PowerDNS and Cloudflare's DNSSEC implementation. Timing attacks could be relevant for DNSSEC signers when online/on-demand signing is supported when the signing operations are not constant-time operations. For signature validation on resolvers this would not be a relevant property however, as no secret keys are processed during validation.

5.3 Cache timing attack example

Implementing constant-time code is hard. Even when the elliptic curve point multiplication itself is implemented in constant time, there could be other operations that could leak timing information that could lead to the compromise of the private key that is used. In this section a practical example of what else could go wrong is illustrated.

Garcia and Brumley [13] discovered in December 2016 that P-256 was vulnerable to cache timing attacks in OpenSSL 1.0.1u, despite that it featured code for constant time scalar multiplications and nonce inversions that were required to generate signatures. The reason that is was vulnerable to cache timing attack was because the nonce inversion (k^{-1}) operation was not set to be a constant time operation by default.

Conditional branches were used in order to calculate the nonce inversion using the Binary Extended Euclidean Algorithm (BEEA) [36], which resulted into different information being loaded into the cache when doing the right-shift operations than the subtraction operations, that were needed to calculate the nonce inversion using the BEEA. This made the nonce inversion vulnerable to cache timing attacks in which these operations can be observed by checking if certain information is in the cache or not.

By using the FLUSH+RELOAD [37] cache timing attack technique they could observe these right-shift and subtraction operations and therefore recover bits of several nonces for the signatures that were generated, which allowed them to recover the private key that was used to generate these signatures by using lattice techniques.

The relevance for DNSSEC would be that DNS server software that support DNSSEC are using software libraries such as OpenSSL and could be vulnerable to such an attack as well when a vulnerable version of OpenSSL is used.

By DNSSEC querying random non-existing subdomains, responses including new signatures could be generated for each subdomain for the authenticated denial of existence feature in DNSSEC, when the online-signer feature is enabled.

When the signature generation is not constant time such a timing attack could be used to recover the private key. After recovering the secret key it would be possible to forge signatures, which could compromise the security that is offered by DNSSEC for a DNS zone.

This nonce inversion method is replaced by a constant-time nonce inversion method that does not have this problem in OpenSSL 1.0.2, 1.1.0 and higher. The C and assembly implementations of Ed25519 are designed to be constant-time and resistant to cache timing attacks [14] and there have not been any successful timing attack against Ed25519 so far.

So recommended is to either use a modernized implementation of ECDSA P-256 that is resistant against timing attacks and can generate unique and hard to predict nonces for each unique message or an Ed25519 implementation that is designed from the ground up to offer such protection, if that would be preferable (e.g. due to trust issues with P-256). The use of outdated and broken implementations of ECDSA P-256 are not recommended however.

6. Performance

The second part of the results concerns the second research question: How does the performance of ECDSA P-256 compare to Ed25519 on different hardware architectures?

Benchmarks have been done on an Intel and ARM processor to represent Intel and ARM based servers and an MIPS processor to represent a MIPS based home router. Some implementations include fast assembly language implementations of Ed25519 or ECDSA P-256 for a processor architecture that is used. These fast assembly language implementations are referred to as optimized implementation in this section.

ECDSA P-256 in different versions of OpenSSL are benchmarked. OpenSSL 0.9.8 is taken as the legacy version, 1.0.1 as the non-optimized version, 1.0.2 as the version that includes the optimized NISTZ256 (also known as ecp_nistz256) implementation of P-256 for x86/x64 and 1.1.0 as the latest stable release branch that also includes the optimized NISTZ256 implementation of P-256 for ARM.

The Ed25519 implementations in Libsodium, OpenSSL and ed25519-donna have been benchmarked. The motivation for including Libsodium and OpenSSL is that they are popular cryptographic libraries that work well on multiple architectures. The motivation for including ed25519-donna is that it is optimized for x64. Note that Ed25519 has been integrated into OpenSSL 1.1.1-dev since commit 04dec1a from 25 May 2017, 15:53 CEST [38] . This version is based on the ref10 reference implementation and it expected to perform worse than an optimized implementation of Ed25519, but it makes sense to include it, because implementers could choose to only use OpenSSL, in which case they would get an optimized ECDSA P-256 implementation, but a non-optimized Ed25519 implementation at the moment. A good reason to only use OpenSSL is to reduce the number of dependencies for a software package and because OpenSSL is widely available on many platforms.

For the MIPS processor, OpenSSL 1.1.x was not available for OpenWrt at the time of doing the benchmarks on the OpenWrt router, which why these versions have been of OpenSSL have been excluded. The other implementations have been cross-compiled using the OpenWrt SDK [39] in order to run them on OpenWrt.

Note that the speed is not only varying because of optimizations through several versions of OpenSSL, but also because of bug fixes and other (minor) changes that could affect the speed.

6.1 Intel

Benchmarking results on the Intel Xeon E3-12xx V2 @2.4Ghz for signature verification and signature generation have been shown in table 1V and 1S and the speed improvements for Ed25519 are shown in table 1CV and 1CS.

6.1.1 Intel: Verifying

Table 1V: Verifications/second of P-256 and Ed25519 on a 2.4Ghz Intel Xeon E3-12xx v2 (Ivy Bridge) running Ubuntu 16.04.4 LTS

	P-256 in OpenSSL 0.98x	P-256 in OpenSSL 1.0.1f	P-256 in OpenSSL 1.0.2e	P-256 in OpenSSL 1.1.0f	Ed25519 in ed25519-donna	Ed25519 in Libsodium 1.0.12	Ed25519 in OpenSSL 1.1.1-dev
Verifications/ second	1968.6	1944.0	6579.4	6345.8	12345.0	5502.3	6043.9
Standard deviation	28.6	31.0	78.1	83.8	139.1	16.3	43.1

Table 1CV: Signature verification speed improvement of Ed25519 compared to P-256 on a 2.4Ghz Intel Xeon E3-12xx v2 (Ivy Bridge) running Ubuntu 16.04.4 LTS

Signature verification speed improvements for Ed25519	Ed25519 in ed25519-donna (=optimized for x86/x64)	Ed25519 in Libsodium 1.0.12	Ed25519 in OpenSSL 1.1.1-dev
P-256 in OpenSSL 0.98x	~6.3X	~2.8X	~3.1X
P-256 in OpenSSL 1.0.1f	~6.4X	~2.8X	~3.1X
P-256 in OpenSSL 1.0.2e (=optimized for x86/x64)	~1.9X	~0.8X	~0.9X
P-256 in OpenSSL 1.1.0f (=optimized for x86/x64)	~1.9X	~0.9X	~1X

As shown in table 1CV, Ed25519 verifies signatures faster when comparing the non-optimized versions of Ed25519 to the non-optimized versions of P-256. E.g. the non-optimized version of Ed25519 in OpenSSL 1.1.1-dev is ~3.1X as fast as the non-optimized version of P-256 in OpenSSL 1.0.1f. Ed25519 is also faster when comparing the optimized version of Ed25519 to the optimized version of P-256. E.g. Ed25519 in ed25519-donna is ~1.9X as fast as P-256 in OpenSSL 1.1.0f. When comparing the non-optimized version of Ed25519 to an optimized version of P-256 in OpenSSL the signature verification speeds are similar to each other. As explained before, such a comparison would make sense, considering that DNSSEC software could rely on cryptographic implementations that are available in OpenSSL.

6.1.2 Intel: Signing

Table 1S: Signatures/second of P-256 and Ed25519 on a 2.4Mhz Intel Xeon E3-12xx v2 (Ivy Bridge) running Ubuntu 16.04.4 LTS

	P-256 in OpenSSL 0.98x	P-256 in OpenSSL 1.0.1f	P-256 in OpenSSL 1.0.2e	P-256 in OpenSSL 1.1.0f	Ed25519 in ed25519-donna	Ed25519 in Libsodium 1.0.12	Ed25519 in OpenSSL 1.1.1-dev
Signatures/second	7720.2	7329.9	15909.8	14905.0	40872.0	14463.6	20123.2
Standard deviation	57.0	71.7	361.3	556.2	294.2	237.9	146.7

Table 1CS: Signature generation speed improvement of Ed25519 compared to P-256 on a 2.4Ghz Intel Xeon E3-12xx v2 (Ivy Bridge) running Ubuntu 16.04.4 LTS

Signature generation speed improvements for Ed25519	**Ed25519 in ed25519-donna (=optimized for x86/x64)**	**Ed25519 in Libsodium 1.0.12**	**Ed25519 in OpenSSL 1.1.1-dev**
P-256 in OpenSSL 0.98x	~5.3X	~1.9X	~2.6X
P-256 in OpenSSL 1.0.1f	~5.6X	~2.0X	~2.7X
P-256 in OpenSSL 1.0.2e (=optimized for x86/x64)	~2.6X	~0.9X	~1.3X
P-256 in OpenSSL 1.1.0f (=optimized for x86/x64)	~2.7X	~1X	~1.4X

As shown in table 1CS, the non-optimized versions of Ed25519 generates signatures faster than the non-optimized versions of P-256. E.g. the non-optimized version of Ed25519 in OpenSSL 1.1.1-dev is ~2.7X as fast as the non-optimized version of P-256 in OpenSSL 1.0.1f. Ed25519 is also faster when comparing the optimized version of Ed25519 to the optimized version of P-256. E.g. Ed25519 in ed25519-donna is ~2.7X as fast as P-256 in OpenSSL 1.1.0f. When comparing the non-optimized version of Ed25519 to an optimized version of P-256 the performance is similar to each other. In which the OpenSSL implementation of Ed25519 is slightly faster.

6.2 ARM

Benchmarking results on the ARM Cortex A-53@1.2Ghz have been shown in table 2V and 2S and the speed improvements for Ed25519 are shown in table 2CV and 2CS. Note that there was no implementation for Ed25519 available that specifically optimizes for ARM at the time of these benchmarks.

6.2.1 ARM: Verifying

Table 2V: Verifications/second of P-256 and Ed25519 on a 1.2GHz ARM Cortex A-53 running Ubuntu MATE 16.04.4 LTS

	P-256 in OpenSSL 0.98x	**P-256 in OpenSSL 1.0.1f**	**P-256 in OpenSSL 1.0.2e**	**P-256 in OpenSSL 1.1.0f**	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**	**Ed25519 in OpenSSL 1.1.1-dev**
Verifications /second	130.5	221.2	272.5	752.9	695.0	531.9	556.4
Standard deviation	14.6	21.5	35.7	66.2	44.3	33.1	30.7

Table 2CV: Signature verification speed improvement of Ed25519 compared to P-256 on a 1.2GHz ARM Cortex A-53 running Ubuntu MATE 16.04.4 LTS

Signature verification speed improvements for Ed25519	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**	**Ed25519 in OpenSSL 1.1.1-dev**
P-256 in OpenSSL 0.98x	~5.3X	~4.1X	~4.3X
P-256 in OpenSSL 1.0.1f	~3.1X	~2.4X	~2.5X
P-256 in OpenSSL 1.0.2e	~2.6X	~2.0X	~2.0X
P-256 in OpenSSL 1.1.0f (=optimized for ARM)	~0.9X	~0.7X	~0.7X

As shown in table 2CV, the 3 Ed25519 implementations have faster signature validations than to the non-optimized version of P-256 in OpenSSL 0.9.8x,1.0.1f and 1.0.2e, but they have a comparable or slightly lower signature verification speed than the optimized P-256 implementation in OpenSSL 1.1.0f

6.2.2 ARM: Signing

Table 2S: Signatures/second of P-256 and Ed25519 on a Cortex A-53 of a Raspberry Pi 3

	P-256 in OpenSSL 0.98x	**P-256 in OpenSSL 1.0.1f**	**P-256 in OpenSSL 1.0.2e**	**P-256 in OpenSSL 1.1.0f**	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**	**Ed25519 in OpenSSL 1.1.1-dev**
Signatures/ second	660.8	988.1	1077.1	1927.2	2179.8	1444.7	1730.8
Standard deviation	56.0	109.1	131.0	159.8	144.6	81.9	105.2

Table 2CS: Signature generation speed improvement of Ed25519 compared to P-256 on a 1.2GHz ARM Cortex A-53 running Ubuntu MATE 16.04.4 LTS

Signature generation speed improvements for Ed25519	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**	**Ed25519 in OpenSSL 1.1.1-dev**
P-256 in OpenSSL 0.98x	~3.3X	~2.2X	~2.6X
P-256 in OpenSSL 1.0.1f	~2.2X	~1.5X	~1.8X
P-256 in OpenSSL 1.0.2e	~2.0X	~1.3X	~1.6X
P-256 in OpenSSL 1.1.0f (=optimized for ARM)	~1.1X	~0.7X	~0.9X

As shown in table 2CS, the 3 Ed25519 implementations have faster signature generation speed than to the non-optimized version of P-256 in OpenSSL 0.9.8x,1.0.1f and 1.0.2e, but they have a comparable or slightly lower signature generation speed than the optimized P-256 implementation in OpenSSL 1.1.0f.

6.3 MIPS

6.3.1 MIPS: Verifying

Table 3V: Verifications/second of P-256 and Ed25519 on a 0.4 Ghz MIPS AR9331 running OpenWrt 15.05.1

	P-256 in OpenSSL 0.98x	**P-256 in OpenSSL 1.0.1f**	**P-256 in OpenSSL 1.0.2e**	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**
Verifications /second	17.0	31.4	35.0	66.5	239.5
Standard deviation	0.2	1.0	0.7	0.5	3.1

Table 3CV: Signature verification speed improvement of Ed25519 compared to P-256 on a 0.4 Ghz MIPS AR9331 running OpenWrt 15.05.1

Signature verification speed improvements for Ed25519	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**
P-256 in OpenSSL 0.98x	~3.9X	~14.1X
P-256 in OpenSSL 1.0.1f	~2.1X	~7.6X
P-256 in OpenSSL 1.0.2e	~1.9X	~6.8X

As shown in table 3CV, Ed25519 has a higher signature verification speed than P-256 in OpenSSL. Verifications for Ed25519 in ed25519-donna were ~1.9 times as fast verifications for P-256 in OpenSSL 1.0.2e. Libsodium was several times faster. Verifications for Ed25519 in Libsodium 1.0.12 were ~6.8 times as fast verifications for P-256 in OpenSSL 1.0.2e

6.3.2 MIPS: Signing

Table 3S: Signatures/second of P-256 and Ed25519 on a 0.4 Ghz MIPS AR9331 running OpenWrt 15.05.1

	P-256 in OpenSSL 0.98x	**P-256 in OpenSSL 1.0.1f**	**P-256 in OpenSSL 1.0.2e**	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**
Verifications /second	83.9	153.4	137.2	198.6	508.7
Standard deviation	0.6	2.8	1.0	12.2	13.4

Table 3CS: Signature generation speed improvement of Ed25519 compared to P-256 on a 0.4 Ghz MIPS AR9331 running OpenWrt 15.05.1

Signature generation speed improvements for Ed25519	**Ed25519 in ed25519-donna**	**Ed25519 in Libsodium 1.0.12**
P-256 in OpenSSL 0.98x	~2.4X	~6.1X
P-256 in OpenSSL 1.0.1f	~1.3X	~3.3X
P-256 in OpenSSL 1.0.2e	~1.4X	~3.7X

As shown in table 3CS, Ed25519 is performing better than P-256 in OpenSSL. Signature generations for Ed25519 in ed25519-donna were ~1.4 times as fast verifications for P-256 in OpenSSL 1.0.2e. Libsodium was several times faster. Verifications for Ed25519 in Libsodium 1.0.12 were ~3.7 times as fast verifications for P-256 in OpenSSL 1.0.2e

7. Discussion

This section reflects on the research results and revisits the research questions.

7.1 Is it worth switching from Ed25519 signatures to P-256 signatures in DNSSEC?

This would depend on the case, as explained in the following subsections.

7.1.1 In order to completely switch from ECDSA P-256 to Ed25519 in DNSSEC signed zones in June 2017?

No, there is not enough support for Ed25519 on resolvers. This would be a disadvantage, as without wide support, you would lose the security advantages DNSSEC provides. The improvements in speed and security that are made to modern implementations of ECDSA P-256 also make that there is less urgency to switch to a new signature algorithm. OpenSSL 1.0.2, 1.1.x or higher are recommended for ECDSA P-256, because it is featuring faster and secure constant-time implementations of ECDSA P-256. OpenSSL 1.0.1 and lower would not be recommended for ECDSA P-256, as these versions are slower, no longer supported and potentially insecure. Another thing that has to be considered is the costs of switching compared to the small amount of improvements of Ed25519 compared to modern implementations of ECDSA P-256.

7.1.2 If you have some domains to experiment with and want to try the latest algorithms in DNSSEC?

Yes, the standardization of Ed25519 in DNSSEC is a good opportunity for early adopters to experiment with Ed25519 in DNSSEC.

7.1.3 If the optimized versions of Ed25519 would be become widely supported everywhere soon?

Then Ed25519 could be a good alternative to ECDSA P-256, because of the design, speed and reputation.

The design could make people worry less about potential bugs in some implementations and allows smaller public key sizes than ECDSA P-256, which could further reduce the bandwidth requirements for DNSSEC (i.e. communicating the public key in DNSKEY records would require half the size of a ECDSA P-256 key). Improved signature generation speed and signature validation speed have as benefit that they could reduce the server load. Ed25519 has as extra benefit that it does not have the same reputation issues that other algorithms in DNSSEC face, such as causing DDoS attacks or potentially being back doored or causing security issues, which could convince more people to implement DNSSEC who currently are not implementing DNSSEC.

7.2 Does Ed25519 offer better security than ECDSA P-256 and and are the differences relevant enough to prefer Ed25519 signatures over P-256 signatures in DNSSEC?

In some cases, yes. For example, in case of the cache timing attack bug for ECDSA P-256 that could lead to key compromise.

In some other cases it still has not been shown that the attacks would be feasible for ECC signatures, but using Ed25519 could give people less to worry about.

Ed25519 has a design that provides more resistance against hash collisions, which is a conservative design that could have benefits in the long term. But currently there's no need to worry about hash collisions for the SHA256 hash that is used for ECDSA P-256 in DNSSEC either.

There are improvements security of ECDSA P-256 as well, such as faster and constant-time implementations to prevent timing attacks in which case it could even be argued that is just as secure as Ed25519.

For NISTZ256 it should also be noted that for early versions of NISTZ256 in OpenSSL there were some bugs [40] [41] that could cause incorrect results in some exceptional cases. These bugs have been fixed later, so upgrading to the latest version of OpenSSL is recommended.

There could still be undiscovered bugs however, but this should be manageable, considering that there could also be many other bugs that are not cryptography related that could be much easier to exploit by hackers.

Summarizing, there is no definitive answer on whether Ed25519 is more secure than ECDSA P-256 or not, as it would depend on the implementation and on how much you trust the curves.

7.3 How does the performance of ECDSA P-256 compare to Ed25519 on different hardware architectures?

On the Intel processor the fastest implementation of Ed25519 that was benchmarked was 2 times as fast as the fastest implementation that was benchmarked for ECDSA P-256 for signature generation and validation, so there would be a speed benefit of using Ed25519 over ECDSA P-256. On the ARM processor the fastest implementation of ECDSA P-256 had approximately the same speed as the fastest implementation of Ed25519. In this case it also be noted that there was no optimized ARM version of Ed25519 available at the time of the benchmarking. In which case there would not be a speed benefit of using Ed25519 over ECDSA P-256.

The optimized implementations of Ed25519 have not been available everywhere. Libsodium 1.0.12 and OpenSSL 1.1.1-dev implementations of Ed25519 that were benchmarked still used the reference implementation that had approximately the same performance as the optimized version of ECDSA P-256 on the Intel and ARM processor.

It is currently uncertain how widely available the optimized version of Ed25519 will become.

Further improvements in the speed of ECDSA P-256 might be possible as well. A NIST sponsored project [42] suggests that it would be possible to make signature verification for ECDSA P-256 even faster than the NISTZ256 implementation when using a variable time implementation. The argumentation that was given for this was that no secret information was processed during signature validation and therefore no protection against timing attacks would be needed. It was not possible to verify this claim about the performance improvements however, as the implementation was not publicly available.

Benchmarks were also done on a MIPS processor. On the MIPS processor the fastest implementation of Ed25519 was ~6.8X as fast for verifying signatures and ~3.7X as fast for generating signatures as the fastest implementation of ECDSA P-256 that was benchmarked. Which would make Ed25519 preferable over ECDSA P-256.

8. Conclusion

The goal was to investigate if it is worth switching from P-256 signatures to Ed25519 signatures in DNSSEC for security and performance reasons.

Several security concerns for ECDSA P-256 have already been addressed in modern implementations of ECDSA P-256, which makes that there's less urgency for a new signature algorithm, while the concern on whether the NIST curves are trustworthy remains. In the benchmarks that were done the optimized implementations of Ed25519 were faster than the optimized and non-optimized versions of ECDSA P-256, but the performance of the non-optimized version of Ed25519 that is still used in some implementations was neck and neck with the optimized version of P-256. If optimized versions Ed25519 would be widely available for DNSSEC soon, then it could be a good alternative for ECDSA P-256. As it would improve upon speed compared to currently available open source implementations of ECDSA P-256 without compromising on security.

9. Bibliography

[1] R. v. Rijswijk-Deij, A. Sperotto and A. Pras, "DNSSEC and its potential for DDoS attacks: a comprehensive measurement study," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014.

[2] G. V. D. Broek, R. v. Rijswijk-Deij, A. Sperotto and A. Pras, "DNSSEC meets real world: dealing with unreachability caused by fragmentation," *IEEE communications magazine,* vol. 52, no. 4, pp. 154--160, 2014.

[3] R. v. Rijswijk-Deij, A. Sperotto and A. Pras, "Making the case for elliptic curves in DNSSEC," *ACM SIGCOMM computer communication review,* vol. 45, no. 5, pp. 13--19, 2015.

[4] R. v. Rijswijk-Deij, M. Jonker, A. Sperotto and A. Pras, "A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements," *IEEE Journal on Selected Areas in Communications,* vol. 34, no. 6, pp. 1877--1888, 2016.

[5] S. Josefsson and I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA), RFC8032,* 2017.

[6] O. Sury and R. Edmonds, *Edwards-Curve Digital Signature Algorithm (EdDSA), RFC8080,* 2017.

[7] O. Surý, *New Curves in DNSSEC,* 2016.

[8] D. J. Bernstein and T. Lange, "Security dangers of the NIST curves," in *Invited talk, International State of the Art Cryptography Workshop, Athens, Greece*, 2013.

[9] P. Hoffman and W. Wijngaards, *Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC, RFC6605,* 2012.

[10] R. v. Rijswijk-Deij, K. Hageman, A. Sperotto and A. Pras, "The Performance Impact of Elliptic Curve Cryptography on DNSSEC Validation," *IEEE/ACM Transactions on Networking,* vol. 25, no. 2, pp. 738-750, April 2017.

[11] M. Nystrom, "Last Call Review of draft-ietf-curdle-dnskey-eddsa-02," 15 06 2017. [Online]. Available: https://datatracker.ietf.org/doc/review-ietf-curdle-dnskey-eddsa-02-secdir-lc-nystrom-2016-12-15/.

[12] D. J. Bernstein and T. Lange, "SafeCurves: choosing safe curves for elliptic-curve cryptography.," [Online]. Available: https://safecurves.cr.yp.to. [Accessed 01 06 2017].

[13] C. P. Garcia and B. B. Brumley, "Constant-time callees with variable-time callers," 2016.

[14] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering,* pp. 1--13, 2012.

[15] D. J. Bernstein and T. Lange, "eBACS: ECRYPT Benchmarking of Cryptographic Systems," [Online]. Available: https://bench.cr.yp.to. [Accessed 01 06 2017].

[16] The OpenSSL Project Authors, "openssl/crypto/ec/asm/," [Online]. Available: https://github.com/openssl/openssl/tree/master/crypto/ec/asm.

[17] O. Kolkman, W. Mekking and R. Gieben, *DNSSEC Operational Practices Version 2, RFC6781,* 2012.

[18] M. Stevens, E. Bursztein, P. Karpman, A. Albertini and Y. Markov, "The first collision for full SHA-1.," *IACR Cryptology ePrint Archive,* vol. 2017, p. 190, 2017.

[19] D. Jonson, 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.

[20] D. J. Bernstein, "[Cfrg] 25519 naming," [Online]. Available: https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html. [Accessed 3 06 2017].

[21] S. Gueron, "#3149: [patch] Fast and side channel protected implementation of the NIST P-256 Elliptic Curve, for x86-64 platforms," 22 10 2013. [Online]. Available: https://rt.openssl.org/Ticket/Display.html?id=3149. [Accessed 01 06 2017].

[22] H. Cantero, S. Peter and S. Bushing, "Console Hacking 2010--PS3 Epic Fail," *27th Chaos Communication Congress (December 2010),* 2010.

[23] T. Pornin, *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), RFC6979,* 2013.

[24] A. Langley, "Make `safe' (EC)DSA nonces the default.," 13 07 2013. [Online]. Available: https://github.com/openssl/openssl/commit/190c615d4398cc6c8b61eb7881d7409314529a75. [Accessed 10 06 2017].

[25] N. A. Howgrave-Graham and N. P. smart, "Lattice attacks on digital signature schemes," *Designs, Codes and Cryptography,* vol. 23, no. 3, pp. 283--290, 2001.

[26] P. Q. Nguyen and I. E. Shparlinski, "The insecurity of the elliptic curve digital signature algorithm with partially known nonces," *Designs, codes and cryptography,* vol. 30, no. 2, pp. 201--217, 2003.

[27] B. Brumley and N. Tuveri, "Remote timing attacks are still practical," *Computer Security--ESORICS 2011,* pp. 355-371, 2011.

[28] Y. Yarom, D. Genkin and N. Heninger, "CacheBleed: A Timing Attack on OpenSSL Constant Time RSA.," in *CHES*, 2016.

[29] S. Gueron and V. Krasnov, "Fast prime field elliptic-curve cryptography with 256-bit primes," *Journal of Cryptographic Engineering,* vol. 5, no. 2, pp. 141--151, 2015.

[30] D. J. Bernstein, T. Lange and R. Niederhagen, "Dual EC: A standardized back door," in *The New Codebreakers*, Springer, 2016.

[31] N. Koblitz and A. Menezes, "A riddle wrapped in an enigma," *IEEE Security \& Privacy,* vol. 14, no. 6, pp. 34--42, 2016.

[32] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Asiacrypt*, 2007.

[33] D. Brown, "Re: [Cfrg] Complete additon for cofactor 1 short Weierstrass curve?," 09 06 2017. [Online]. Available: https://www.ietf.org/mail-archive/web/cfrg/current/msg05651.html.

[34] J. Renes, C. Costello and L. Batina, "Complete addition formulas for prime order elliptic curves.," in *EUROCRYPT (1)*, 2016.

[35] T. Izu and T. Takagi, "Exceptional procedure attack on elliptic curve cryptosystems," in *International Workshop on Public Key Cryptography*, 2003.

[36] A. J. Menezes, P. C. V. Oorschot and S. A. Vanstone, "Chapter 14.4.3," in *Handbook of applied cryptography*, CRC press, 1996.

[37] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack.," in *USENIX Security Symposium*, 2014, pp. 719--732.

[38] S. Henson, "Clear sensitive data in ED25519_sign," 25 05 2017. [Online]. Available: https://github.com/openssl/openssl/commit/04dec1ab34df70c1588d42cc394e8fa8b5f3191c. [Accessed 01 06 2017].

[39] OpenWrt, "OpenWrt-SDK-15.05.1-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64," [Online]. Available: https://downloads.openwrt.org/chaos_calmer/15.05.1/ar71xx/generic/OpenWrt-SDK-15.05.1-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64.tar.bz2. [Accessed 20 05 2017].

[40] J. Sun, "[openssl-dev] [openssl.org #4284] Bug in nistz256 assembly code.," [Online]. Available: https://mta.openssl.org/pipermail/openssl-dev/2016-February/004701.html. [Accessed 03 06 2017].

[41] B. Smith, "[openssl-dev] [openssl.org #4621] BUG: nistz256 point addition check for a = +/-b doesn't work for unreduced values," 04 06 2017. [Online]. Available: https://mta.openssl.org/pipermail/openssl-dev/2016-July/007963.html.

[42] M. Adelier, "Efficient and Secure Elliptic Curve Cryptography Implementation of Curve P-256".

[43] A. Barenghi and G. Pelosi, "A Note on Fault Attacks Against Deterministic Signature Schemes," in *International Workshop on Security*, 2016.