ECDSA Security in Bitcoin and Ethereum: a Research Survey

Hartwig Mayer
{hartwig.mayer}@coinfabrik.com

CoinFabrik

Revised June 28, 2016

Abstract

This survey discusses the security level of the signature scheme implemented in Bitcoin and Ethereum.

1 Introduction

To authorize payments, Bitcoin and Ethereum use the Elliptic Curve Digital Signature Algorithm (ECDSA). The two ingredients for this algorithm are an elliptic curve and a cryptographic hash function. Both Bitcoin and Ethereum work with the elliptic curve SECP256K1 which is recommended by the Standards for Efficient Cryptography Group (see [SEC10]) and the hash function SHA256 (see [SHA13]).

Before we examine the security level of ECDSA with the above curve and hash function, we would like to make some general comments. Firstly, mathematical complexity does not necessarily guarantee the security of a cryptographic algorithm when it is implemented in a real-world situation. Implementation can dramatically lower the security level: for example, in 2011, the Playstation ECDSA security break took advantage of low randomness in the signing procedure. In section 3 of this article we will come back to some implementation issues in connection with SECP256K1.

All elliptic curve cryptography is based on the belief that the discrete logarithm problem on an elliptic curve (see 1.1 below) is very difficult to solve. Although these discrete logarithm problems are considered to be of exponential complexity, there is currently no proof that they are not solvable in polynomial time. Note that such proof would imply that $P \neq NP$ which would answer one of the famous open questions in computer science. To date, only a very specific class of elliptic curves, those defined over binary fields, is thought by some experts to allow subexponential running time (cf. [GG16]).

Our focus here is mainly to collect the known attack strategies for the discrete logarithm problem on elliptic curves. We will cross check these strategies against the SECP256K1 curve in section 2. In section 3, we discuss some implementation issues, and in section 4, other recommended elliptic curves and signature algorithms. At the end of this article one can find a short summary of our analysis.

1.1. *The Discrete Logarithm Problem on Elliptic curves and ECDSA*. We begin by recalling some basic notions from mathematics. An elliptic curve (in short affine Weierstrass form) E over a finite field $\mathbb{F}_q$, $q = p^n$ a prime power, (e.g. if $n = 1$ then $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$), is the set of solutions of an equation

$$E \ : \ y^2 = x^3 + Ax + B \qquad (A, B \in \mathbb{F}_q),$$

that is

$$E(\mathbb{F}_q) := \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q \mid y^2 = x^3 + Ax + B\}.$$

On an elliptic curve E we have a group operation as we have for example the operation ' $+$ ' on the integers $\mathbb{Z}$ (see e.g. [Was08]). The archetype of the discrete logarithm problem (DLP) is formulated in the multiplicative group $(\mathbb{F}_p^\times, \cdot)$, namely: given $g, h \in \mathbb{F}_p^\times$, find $k \in \mathbb{Z}/(p-1)\mathbb{Z}$ such that $g^k = h \mod (p)$. Formulating the same computational task for elliptic curves and using the convention to write ' $+$ ' instead of ' $\cdot$ ' results in the Elliptic Curve Discrete Logarithm Problem (ECDLP): *Given two points $P, Q \in E(\mathbb{F}_q)$ on E, find an integer a, if it exists, such that the equation $Q = aP$ holds. Or, using the logarithm notation in an intuitive way, compute $a = \log_P(Q)$ on E.*
The signing algorithm ECDSA (Elliptic Curve Digital Signature Algorithm) uses the integer a as the private key. Let us quickly outline how it works: We suppose that a user U wants to sign a message m to authorize a payment to a merchant M. Given (public) is an elliptic curve $E/\mathbb{F}_q$ over a finite field $\mathbb{F}_q$, a base point $P \in E(\mathbb{F}_q)$ (the base of our logarithm) of order r, and a hash function H (its values will be considered modulo r) which is preimage and collision resistant. Then:

- Private key: U chooses a secrete integer a mod r , i.e. $a \in \mathbb{Z}/r\mathbb{Z}$.
- Public key: U computes $Q = aP$ and publishes Q. (In Bitcoin the address of the user U is derived from this data).

Then the actual algorithm goes as follows:

Signing:

(a) U chooses a *random* (each time a different one!) number $1 \leq k < r$ and computes $R = (x_R, y_R) = kP \in E(\mathbb{F}_q)$.
(b) Then U computes $s = k^{-1}(H(m) + ax_R)$ in $\mathbb{Z}/r\mathbb{Z}$ (well-defined). U's signature is then

$$(x_R, s).$$

Verifying

(a) M computes $u_1 = s^{-1}H(m)$ and $u_2 = s^{-1}x_R$ in $\mathbb{Z}/r\mathbb{Z}$, and then $V = (x_V, y_V) = u_1P + u_2Q$ on E.
(b) M verifies the signature of user U by checking if $x_R = x_V$ in $\mathbb{Z}/p\mathbb{Z}$ holds.

If someone could compute discrete logarithms very fast then she would be able to calculate the private key a from P and Q, and to sign payments in the name of user U. More about the security of signature system ECDSA can be found in ([Bro05], and [HMV04], 4.4.1, p. 184).

2 Security of ECDLP with SECP256K1

There are several different standards to secure Elliptic Curve Cryptography (ECC) by making ECDLP infeasible, e.g.: ANSI X9.63 (1999), IEEE P1363 (2000), SEC 2 (2000), NIST FIPS 186-2 (2000), Brainpool (2005), NSA Suite B (2005), Certicom SEC 2 v2 (2010), OSCCA SM2 (2010), NIST FIPS 186-4 (2013), Safecurves (2013). We follow the line of Safecurves (see [BL13]) as it is one of the most recent and most strict standards which also covers related implementation issues.

Set-up: *Let $E/\mathbb{F}_q$ be an elliptic curve over $\mathbb{F}_q$ with $q = p^n$. We assume that the order of the group $E(\mathbb{F}_q)$ is*

$$\#E(\mathbb{F}_q) = f \cdot r$$

for some prime r and arbitrary $f \in \mathbb{N}$.

All standards share the condition that the co-factor f equals 1 or is at least very small (safe against subgroup attacks like Pohlig-Hellman) and work over prime fields, i.e., n=1 (no subfield curves). This check will therefore not enter our discussion. The choice of the prime p is also more about efficiency, besides it has to be big of course. Some primes provide a faster field arithmetic in $\mathbb{F}_q$ than others, but this will not be discussed here.
To measure the security level of ECDLP for a specific curve E one has to know the (average) running time of the possible attacks. The most naive one would be to calculate all multiples of P in the group $E(\mathbb{F}_q)$ until we find Q. This would take us in the worst case r group operations and in average $r/2$. So to protect the private key against this attack, one should choose r so that $r/2$ is large enough! Safecurves recommends that the running time is bigger than 2^{100} group operations on E. In subsection 2.2 the known attack strategies and their implied standards are applied to SECP256K1.

2.1. *The elliptic curve* SECP256K1. This elliptic curve domain is recommended by the Standards for Efficient Cryptography Group. See their website [SEC10] to get more informations. It is designed for a 256-bit prime.
The elliptic curve SECP256K1 is defined over the finite field $\mathbb{F}_p$ with

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

a generalized Mersenne prime number (faster field arithmetic). The defining equation of this elliptic curve is

$$E: \ y^2 = x^3 + 7.$$

Using the computer algebra system PARI, one easily verifies that

$$\#E(\mathbb{F}_p) = 1 \cdot r$$

with prime r equal to

$$\begin{aligned} r =& 2^{256} - 2^{128} - 2^{126} - 2^{122} - 2^{120} - 2^{118} - 2^{116} - 2^{112} - 2^{109} - 2^{105} - 2^{104} - 2^{100} \\ & - 2^{99} - 2^{96} - 2^{94} - 2^{92} - 2^{87} - 2^{85} - 2^{84} - 2^{82} - 2^{81} - 2^{80} - 2^{78} - 2^{76} - 2^{75} - 2^{74} \\ & - 2^{73} - 2^{72} - 2^{71} - 2^{70} - 2^{66} - 2^{62} - 2^{53} - 2^{51} - 2^{50} - 2^{48} - 2^{47} - 2^{45} - 2^{40} - 2^{38} \\ & - 2^{37} - 2^{36} - 2^{33} - 2^{32} - 2^{29} - 2^{27} - 2^{26} - 2^{25} - 2^{24} - 2^{23} - 2^{22} - 2^{19} - 2^{16} - 2^{15} \\ & - 2^{13} - 2^{12} - 2^{11} - 2^{10} - 2^9 - 2^7 - 2^5 - 2^4 - 2^3 - 2^2 - 2^1 - 1. \end{aligned}$$

They also give a base point P, but any point is of order r.

2.2. *Attacks and security standards*: Here we check the elliptic curve SECP256K1 against the security standards of Safecurves. It passes the check if the corresponding attack is not applicable or if r is big enough such that a too long running time is guaranteed.

Pollard's rho method: Pollard's rho method in combination with automorphisms on E produces so far the best results. This method has also the advantage that it is suitable to parallelise the attack on more than one processor (see next chapter on multiple ECDLPs). An up-to-date overview about the running times of these type of algorithms can be found in [GG16], sections 4 and 5.

Algorithm	**Conditions**	**Running Time**	**Check**
Pollard's rho method	no	$(\sqrt{\pi/2}+o(1))\sqrt{r}$	✓
Pollard's ρ method with automorphisms	automorphism with average orbit size l	$(\sqrt{\pi/2}+o(1))\frac{\sqrt{r}}{\sqrt{l}}$	✓

Table 1: Pollard's rho algorithms for SECP256K1

The general theory of elliptic curves tells us that there are six automorphisms on SECP256K1 (as the so-called j-invariant of this curve is zero). And we do actually know that the automorphism group is generated by the automorphism $\psi : E \longrightarrow E$, $(x, y) \mapsto (\zeta_6 x, -y)$, where ζ_6 is a primitive 6th root of unity in $\mathbb{F}_p$ (see [BHH+14]). So in the worst case, the number l in Table 1 would be 6, and we still had $\sqrt{\pi r/12} > 2^{100}$ saying that SECP256K1 is safe against this attack. But we will come back to these automorphisms in the context of speedups in scalar multiplication of points (see below) and twist attacks in subsection 3.2.
Finally, we mention that the probability that this method terminates earlier than the average running time is negligible.

Transfer methods: The idea of these attacks are to reduce efficiently an ECDLP into a discrete logarithm problem in a different group, e.g. $\mathbb{F}_{p^m}^{\times}$ (MOV, Frey-Rück) or in a so-called Jacobian of an hyperelliptic curve (GHS). There one could use an index calculus to solve the problem which has subexponential running time.

Algorithm	**Conditions**	**Running Time**	**Check**
ASSS attack	$r = p$	DLP in $\mathbb{F}_p^+$ (easy!)	✓
MOV attack, Frey-Rück attack	$r\|(p^m - 1)$ for small $m > 0$	DLP in $\mathbb{F}_{p^m}^{\times}$	✓
GHS attack (Weil descent)	strict conditions; at least $n > 2$	faster than Pollard's rho method	✓

Table 2: Transfer algorithms for SECP256K1

Obviously, the ASSS and the GHS attacks do not apply ($r \neq p$ and $n = 1$ in our case). So we are left to check the MOV and Frey-Rück attacks. Without going into the details, the integer m, called *embedding degree*, ensures in some sense that the problem is still complicated enough after the reduction step. The reduction leads

actually to subexponential running time but usually in a much bigger group. To be secure against these attacks Safecurve and Brainpool suggest $m > \frac{r-1}{100}$. SEC1, X9.62, P1363 suggest $m > 20$, $m > 30$ respectively. The curve SECP256K1 satisfies the much stronger condition of Safecurves with $m = \frac{r-1}{6}$.
There is a big improvement on DLP in finite fields, but so far only for some very specific fields exists already quasi-polynomial running time (see [BGJT14]). This does not apply (yet) to prime fields with a large prime. But any progress in this field will affect the transfer methods.

CM elliptic curves: Elliptic curves with a small complex multiplication field discriminant are known to give rise to efficient endomorphism which can be used to accelerate scalar multiplication. This idea was introduced by Gallant, Lambert, and Vanstone (GLV) in [GLV01].

Algorithm	**Conditions**	**Running Time**	**Check**
GLV type	small CM discriminant	speedup of $0,7$	—

Table 3: Efficient endomorphisms for SECP256K1

Safecurves imposes on the CM field discriminant D of an elliptic curve the condition $|D| > 2^{100}$. The discriminant of SECP256K1 is -3 and hence fails the check. Brainpool suggests a less strong bound for which it still fails. Actually, an efficient endomorphism is for example in this case the automorphism ψ mentioned above.
In [SEC10], p. 4, it is explained that the curve SECP256K1 is designed on purpose this way to have a more efficient implementation property. We have seen in 1.1 that to sign and to verify a message one has to compute a multiple of a point. This makes the signing/verification process faster while risking *only* a kind of well understood speedup by a factor of 0.7. So it is arguable whether it is an performance advantage or a lowering of security (by unexpected future improvements). Recent progress indicates anyway that the restriction only to elliptic curves with small CM field discriminant is not really justified anymore (see [GLS09] and [LS14]). Nevertheless, because of the long security history of elliptic curves with small CM field discriminant (especially in pairing-based cyptrography) Safecurves is choosing this very strict bound while saying that there is no evidence at the time of a serious problem.

2.3. *Some further comments.* There is a long list of other attack strategies (most notably the summation polynomial approach suggested by A. Semaev) or refinements of the one mentioned here. As said in the introduction the controversail hope is that these ideas might lead to a subexponetial running time in specific cases. We refer the reader to [GG16] to find details. The authors mention in loc. cit. 10.2. that even in the case an algorithm could be found they would be far from an efficient implementation (actual time, huge storage size).
To get a feeling for the size of the numbers appearing here, let us do a speculative calculation (no guarantee for sensible values!). In how many days we can solve an ECDLP on the elliptic curve SECP256K1 with the actual processor technology? Suppose the execution of one group operation takes 10^{-9} seconds. Then, using the above attack strategies, it would take us

$$10^{-9} \times \sqrt{\pi r/12} \times 0.7 \approx 38.5 \cdot 10^{20} \text{ years.}$$

3 ECDSA in practice

Let us consider some issues appearing when implementing ECDSA. There might arise some vulnerabilities during the curve and key generation or signature generation and verification processes. We survey only problems which are related to the choice of an elliptic curve. General issues which occur while implementing, as for example not checking whether a point is the point at infinity (see here for a report of S. Lerner on this topic), we do not touch here.
The very first vulnerability could be manipulation: a suggested safe curve could have a backdoor insecurity. Bitcoin and Ethereum work with one fixed curve – the SECP256K1 – and only generates private and public keys. According to Safecurves the elliptic curve SECP256K1 can be considered as somewhat 'rigid' meaning that almost all parameters are transparent to the public and hence can be assumed to be not generated in order to be weak. More on manipulation of standardized curves can be read in [BCC+15]. Now we turn to concrete implementation issues:

3.1. *Ladders.* The scalar multiplication of a point P on an elliptic curve E is often used in ECDSA – for example for the public key generation. The so-called Montgomery ladder is a fast and simple algorithm to do this computation in constant time. To implement this ladder the elliptic curve has to be of a specific shape. The curve SECP256K1 does not allow the Montgomery ladder implementation. As a consequence, besides the simplicity and efficiency, SECP256K1 might leak informations for side channel attacks as the time for some computations are not constant. This lead already to successful key extraction and is reflected in libsecp256k1 implementation kit (see [GPP+16]). The author does not know whether this implementation is fast and simple. The Brier-Joye ladder could be applied but slows down the computation too much. Finally, Safecurves recommends a single-coordinate ladder as the Montgomery one, as it makes it easier to implement a protection against the attack we discuss next.

3.2. *Twist security.* The invalid-curve attack could lead to a serious vulnerability for SECP256K1 as pointed out in [BHH+14]. Therefore the attacker uses a similar elliptic curve – a twist of the original curve – and just pretends to use the original curve. If the twist is insecure in the sense of section 2.2 and the implementation does not check whether the point suggested by the attacker does lie on the original curve, he has very good chances to extract the private key after some inquiries. Now, the standard quadratic twist of SECP256K1 is also a safe curve (the cardinality of the group has a 220-bit prime; see [BL13]), but the larger automorphism group leads to four more twists. The biggest prime divisors of their order are 133, 188, 135, and 161 but do have also smaller prime factors making the attack probably more feasible (see [BHH+14], p. 4). A curve which is not twist secure needs to check during the signature and verification procedure whether the points are really on the curve. This is doable but makes the implementation less efficient and less secure. For more details about this attack we refer to [FLRV08] where one can find a computational example with SECP256K1.

3.3. *Completeness & Indistinguishability.* The formulas for the addition and scalar multiplication on an elliptic curve might slightly change for some specific points on the curve. An implementation which does not take care of these exceptions produces errors. Completeness refers to curves having no exceptional cases. The scalar multiplication on the curve SECP256K1 is not complete.

The representation of a point on an elliptic curve usually are distinguishable from a randomly produced string. In order to disguise the appearance of points on elliptic curves there are some strategies available (cf. [BL13]). These constructions are not available for SECP256K1.

3.4. *Multiple ECDLP.* In Bitcoin the public key of any user is generated from SECP256K1, their private keys, and the base point P. The situation looks as follows: Assume we have L users with public keys $Q_i = a_i P \in E(\mathbb{F}_p)$, $1 \leq i \leq L$. If someone wants to find their private keys, she has to solve the following discrete logarithm problems:

$$Q_i = a_i P \qquad (1 \leq i \leq L).$$

Do we have to solve L times a discrete logarithm problem? Or can take advantage of the fact that the ECDLPs takes place on the same elliptic curve with the same base point P? So far there is no algorithm known which finds the solution for one instance faster (see [GG16], p. 4). But using an extended version of Pollard's rho method, once one is found this accelerates finding the other solutions progressively. If for example $L < r^{1/4}$, r is the size of our group, Kuhn and Struik showed that one needs in average $\sqrt{2rL}$ group operations. The extended algorithm of Pollard's rho presented in [HMV04], p. 164, lowers the running time, after finding the first instance, for the second ECDLP by 50 %, the third then by 37%, and so on. So one has to ensure that to find one instance is infeasible which SECP256K1 does.

4 Open discussion

Up to now our focus was on the discrete logarithm problem and the ECDSA scheme applied to the curve SECP256K1 which Bitcoin and Ethereum uses. Let us look a bit outside this frame by surveying some other (standardized) interesting curves and signature schemes frequently discussed as good alternatives.

4.1. *Other (standardized) curves.* A very profound discussion and security check of most of the standardized curves can be found on the website [BL13] of Bernstein and Lange. For example, the NIST P-256 curve suggested by the National Institute of Standards and Technology is graded to be not safe in almost the same categories as the SECP256K1 (3.1 and 3.3 in our discussion). The NIST P-256 passes the CM field discriminant check but fails the rigidity test as seeds for the curve generations are not explained and might have a backdoor insecurity (in 2013 there was the rumor that NSA was generating weak curves). The elliptic curve Curve25519 proposed by D. J. Bernstein in 2006 is the most discussed alternative in formus like bitcointalk or reddit. IETF has standardized this curve and Ripple is considering to support it (see here). The curve Curve25519 passes all the tests we have discussed here (see [BL13]) and has also no extra automorphisms (its j-invariant is $\neq 0, 1728$) and hence lowers possible problems. Curve25519 is used in combination with the signing scheme of Schnorr (see Ed25519 in 4.2). Also the curve E-521 introduced by Bernstein-Lange, Hamburg, Aranha-Barreto-Pereira-Ricardini (independently) in 2013 passes all the security checks of Safecurves but appears less in the discussions in the forums. The most recent contribution is from Hamburg with the curve Ed448-Goldilocks which is also graded to be safe in all catergories.

4.2. *Schnorr algorithm vs ECDSA*. A different signing system which is often discussed as an alternative to the ECDSA is the one introduced by Schnorr. First note that both are theoretically safe signature algorithms. According to A. Back (the founder of Hashcash; check here), the advantages of Schnorr's algorithm are: simpler to securely implement, more forgiving (based on dynamical hash function), clearer security proofs, less constraints on hash function (has not to be collision resistant). The most famous improper implementation of ECDSA happened Sony Playstation. As reported in [BHH+14], poor randomness support during signature generation in the Bitcoin system (the choice of k in section 1.1) already allowed attackers to steal money from clients. The RFC deterministic ECDSA might prevent this problem (see here). Lastly, another advantage is the possibility to use threshold signature – a useful alternative to multi-signature schemes.
The most prominent implementation of a version of Schnorr's algorithm is with the twisted Edward curve Ed25519. This curve is birationally equivalent to Curve25519, in other words, it is almost the curve Curve25519 and can be considered as safe as this one. It shows very fast and secure performances (for more details see [BDL+11]).
There are of course other signature schemes, like ElGamal, and other topics as for example malleability of signatures, but we do not discuss them here.

5 Summary

Let us summarize the essential critical points the choice of SECP256K1 has to our knowledge:

Pure mathematical vulnerabilities:

- Pollard's rho method: Due to the existence of extra automorphisms there is a speedup of Pollard's rho attack but only to a small factor. See section 2.2.
- CM field discriminant: The CM field discriminant of SECP256K1 is small. Brainpool and Safecurves require this invariant to be big to prevent speedups for scalar multiplication. But there is no evidence that this causes a serious problem. See section 2.2.

Implementation related vulnerabilities:

- Ladder: No ladder implementation for scalar multiplication available. This causes no constant time computations which might leak information to side channeling attacks. See section 3.1.
- Twist security: The existence of four (besides the identity and the negation) automorphisms could lead to a serious vulnerability if the signing implementation does not check whether the points are lying on the curve (invalid curve attack). See section 3.2.
- Rigidity: Scalar multiplication has no uniform formulas making the implementation more vulnerable to errors. See section 3.3.
- Indistinguishability: No intrinsic construction available to disguise points on the curve as random strings. See section 3.3.

A vulnerability of ECDSA signature scheme (independent of SECP256K1):

- Randomness: An implementation which does not guarantee enough randomness during the signature process allows an attacker to recover the private key of a user. See section 4.2.

The curve Curve25519 for example has no extra automorphisms which prevent speedups in Pollard rho and eventual invalid curve attacks. It shows at the same time very efficient performances.

References

[BCC+15] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooij, T. Lange, and C. Niederhagen, R. van Vredendaal, *How to manipulate curve standards: a white paper for the black hat*, Preprint. `https://bada55.cr.yp.to/bada55-20150927.pdf` (2015).

[BDL+11] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, *High-speed high-security signatures.*, Cryptographic hardware and embedded systems, Springer, 2011, pp. 124–142.

[BGJT14] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, *A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic*, LNCS (2014), 1–16.

[BHH+14] J.W. Bos, J.A. Halderman, N. Heninger, M Moore, J. Naehrig, and E. Wustrow, *Elliptic Curve Cryptography in Practice*, Lecture Notes in Computer Science **8437** (2014), 157 – 175.

[BL13] D. J. Bernstein and T. Lange, *Safecurves: choosing safe curves for elliptic-curve cryptography*, `http://safecurves.cr.yo.to` (2013).

[Bro05] D. Brown, *On the Provable Security of ECDSA*, Advances in Elliptic Curve Cryptography, London Mathematical Society. Lecture Note Series, vol. 317, Cambridge University Press, Cambridge, 2005, pp. 151–182.

[FLRV08] P. A. Fouque, R. Lercier, D. Real, and F. Valette, *Fault attack on elliptic curves with Montgomery ladder*, FDTC 2008, IEEE-CS (2008).

[GG16] S. Galbraith and P. Gaudry, *Recent progress on the elliptic curve discrete logarithm problem*, Designs, Codes and Cryptography **78** (2016), 51–72.

[GLS09] S. Galbraith, X. Lin, and M. Scott, *Endomorphisms for faster elliptic curve cryptography on a large class of curves.*, Advances in cryptology – EUROCRYPT 2009. Proceedings, Berlin: Springer, 2009, pp. 518–535 (English).

[GLV01] R .P. Gallant, R. J. Lambert, and S. A. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms.*, In J. Kilian (eds.), CRYPT 2001., Berlin: Springer, 2001, pp. 190–200 (English).

[GPP+16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom, *Ecdsa key extraction from mobile devices via nonintrusive physical side channels*, Cryptology ePrint Archive, Report 2016/230, 2016, `http://eprint.iacr.org/`.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.

[LS14] P. Longa and F. Sica, *Four-dimensional Gallant-Lambert-Vanstone scalar multiplication.*, J. Cryptology **27** (2014), no. 2, 248–283 (English).

[SEC10] *Certicom Research. SEC 2: Recommended elliptic curve domain parameters, version 2.0.*, `http://www.secg.org/sec2--v2.pdf`.

[SHA13] FIPS 180-4*: Secure Hash Standards, Federal Information Processing Standards Publication 180-4*, National Institute of Standards and Technology (2013), `http://dx.doi.org/10.6028/NIST.FIPS.180--4`.

[Was08] L. C. Washington, *Elliptic Curves: Number Theory and Cryptography*, Discrete Mathematics and Its Applications, Chapman and Hall/CRC, 2008.