Elliptic Curves and Their Usage in Cryptography From a Programmer's Perspective

Vojtěch Fiala (xfiala61@stud.fit.vutbr.cz)
BUT FIT

***Abstract*—This paper explores elliptic curves from a practical perspective. Instead of delving too deeply into the underlying mathematical background, this work focuses on the practical use of the curves and provides further references for the problems mentioned. First of all, elliptic curves are introduced, including the operations that we can compute with them. Afterwards, their usage in certain cryptography situations is explained.The main focus of this paper is on certain issues that may arise during implementation. This includes performance differences between different curves, implementing an effective scalar multiplication, generating an appropriate public key, and validating given points used in elliptic curve operations. The final part of this paper focuses on attacks on elliptic curves. It presents these attacks in the context of the mentioned problems.**

I. Introduction

Elliptic curves have become an essential part of modern cryptography. Due to their properties, they are an interesting method for the fulfillment of cryptography requirements. They work on the basis of the discrete logarithm problem. To illustrate how popular they are, here are a few examples of where they are employed – cryptocurrencies such as Bitcoin [1] or Ethereum [2], securing communication over the Internet with protocols such as SSL [3] or TLS [4] or in many smart cards [5].

The goal of this paper is to give an overview of what elliptic curves are, what operations do they support, how we can use them in cryptography and what problems that entails. Instead of an in-depth mathematical analysis, this paper will focus on things from the perspective of a programmer who implements elliptic curves as a part of his program. This paper will explore issues that the programmer may want to take into consideration during implementation such as the concept of scalar multiplication and its problems, or selecting the correct curve to suit the programmer's needs.

The first part of this paper will be an introduction to the concept of elliptic curves and an explanation of what operations they support. This will be followed by an introduction to how elliptic curves are used to solve standard cryptographic problems. The second part will focus on what are the most common problems encountered in implementing elliptic curve cryptography and what to avoid. In the last part, attacks on elliptic curves will be presented.

A. Contributions

The contributions of this paper are as follows:

1) The paper explains elliptic curves from the perspective of practical use in the real world.
2) This paper presents known problems with actual implementations of elliptic curve cryptography and practices to consider and avoid when implementing an algorithm using elliptic curves.
3) The paper presents attacks on elliptic curves in the context of the explained problems.

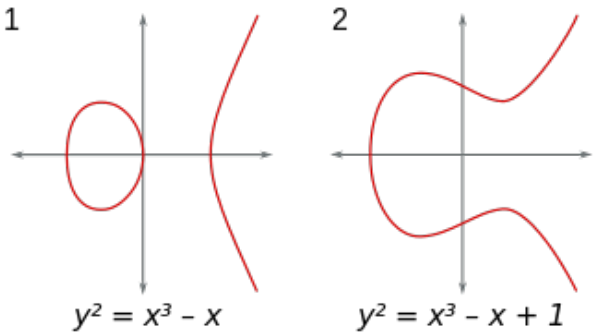

Fig. 1: Example of elliptic curves[1]

II. Background

This section will provide an elementary explanation of elliptic curves and their functionality.

A. Elliptic Curves

Elliptic curves in cryptography are mathematical equations defined over a field of integers modulo p. The purpose of this paper is not to explain the mathematical background, but rather to provide information on their use in cryptography. For more information on the mathematical background, please refer to [6]. From a practical standpoint, without delving too deeply into the mathematical background, it can be stated that an elliptic curve has a form that satisfies the following equation [6]:

$$y^2 = x^3 + ax + b$$

For all a, b parameters belonging to the integer field mod p, the following condition must be satisfied [6]:

$$4a^3 + 27b^2 \not\equiv 0 (mod\ p)$$

The equations below serve as an example of two curves that satisfy the given properties:

$$y^2 = x^3 - x + 1$$

[1]Image taken from https://en.wikipedia.org/wiki/Modular_elliptic_curve

$$y^2 = x^3 - x$$

The visualization of the curves is presented in Figure 1.

B. Operations on Elliptic Curves

Now that it's been established what elliptic curves are, we can discuss the operations that can be performed on them.

Addition

This subsection is based on the explanation provided in [1]. When performing addition over two points A and B on an elliptic curve, there are four possible scenarios that may arise:

- Identity: $B = \infty \Rightarrow A + B = B + A = A$. Infinity is a unique point that functions as the identity element in the context of elliptic curves.
- Negatives: $A = (x, y); B = (x, -y) \Rightarrow A + B = \infty$. In the context of elliptic curves, the negative of a point is a point with the same x coordinate and the opposite y coordinate. The addition of these two points results in the identity element.
- Doubling: $A = (x_1, y_1); B = (x_1, y_1) \Leftrightarrow A = B \Rightarrow A+B = (x_2, y_2)$ where:

$$x_2 = \left(\frac{3 \cdot {x_1}^2 + a}{2 \cdot y_1}\right)^2 - 2 \cdot x_1$$

$$y_2 = \left(\frac{3 \cdot {x_1}^2 + a}{2 \cdot y_1}\right) \cdot (x_1 - x_2) - y_1$$

- Addition: $A = (x_1, y_1); B = (x_2, y_2)$ where $A \neq B \ \& \ A \neq -B \Rightarrow A + B = (x_3, y_3)$ where:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \cdot (x_1 - x_3) - y_1$$

The concept of addition can be illustrated geometrically using the elliptic curve. When adding 2 distinct points, P and Q (where $P \neq Q \ \& \ P \neq -Q$), connect them geometrically using a straight line. Next, extend the line until it intersects the elliptic curve. This point of intersection represents the negative of the result, so to obtain the actual result, multiply the y axis by -1. Figure 2 illustrates the process of geometric addition.

Similarly, a geometric approach can be used to double a point, which means that $P = Q$ and therefore $P + Q \Leftrightarrow 2 \cdot P$. Instead of connecting two points, a tangent is drawn until it intersects the elliptic curve, as described in the previous addition method. The intersection point is the negative of the result and we compute the resulting point by multiplying the y axis value by -1, as before. Figure 3 illustrates this approach.

[2]Image taken from https://www.educative.io/answers/what-is-elliptic-curve-cryptography

[3]Image taken from https://www.educative.io/answers/what-is-elliptic-curve-cryptography

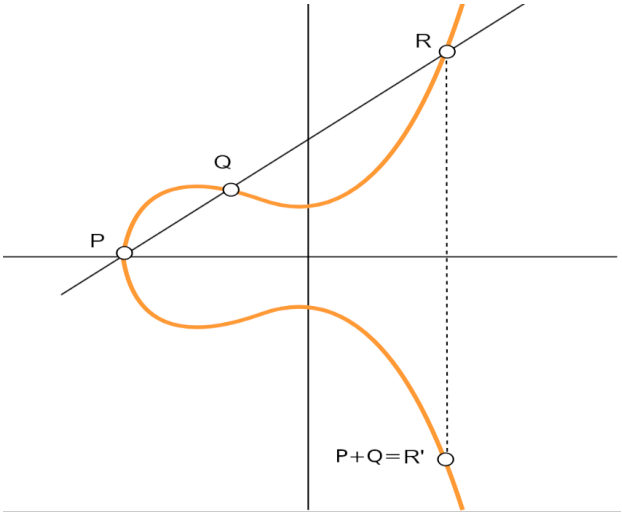

Fig. 2: Geometric addition on an elliptic curve[2]

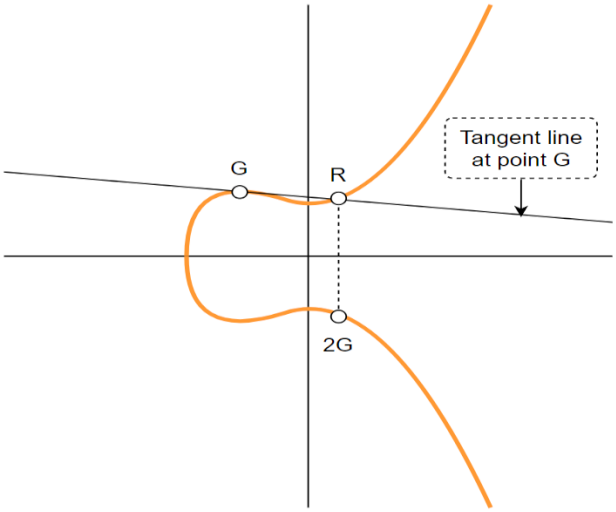

Fig. 3: Geometric doubling of a point on an elliptic curve[3]

Multiplication

The concept of multiplying a point by a constant on an elliptic curve builds upon the previously explained addition. This involves adding the point to itself repeatedly using the addition method explained earlier. However, doing it in such a straightforward way may not be the most effective method. Usually, multiplication employs algorithms such as Double and Add [7] to increase performance.

C. Usage of Elliptic Curves in Cryptography

Three types of cryptography problems will be presented:

- Key exchange
- Signature generation/verification
- Encryption

All of these problem can be solved using the elliptic curves. Elliptic Curve cryptography, and by extension all of the presented algorithms, implement an asymmetric private/public key pair.

Key Exchange – ECDH

The Elliptic Curve Diffie-Hellman (ECDH) algorithm [8], as the name suggests, is based on the standard Diffie-Hellman secret key exchange over an insecure channel. The parties communicating will be referred to as A and B.

Before the algorithm begins, both parties exchange information about the elliptic curve to be used, including a specific generator point G and the range n which limits the maximum public key value.

The algorithm works as follows:

1) Party A chooses a random integer a in range $< 2, n-1 >$ which works as a private key and calculates P which works as a public key:
$$P = a \cdot G$$
P is then sent to party B.
2) Party B similarly chooses a random integer b in range $< 2, n-1 >$ as its private key and calculates its public key Q:
$$Q = b \cdot G$$
Q is then sent to party A.
3) Both parties compute the shared symmetric key S:
$$S = a \cdot Q = b \cdot P = a \cdot b \cdot G = b \cdot a \cdot G$$
The key may then used to encrypt following communication.

Signature generation/verification – ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) [9] is utilized to generate and verify signatures for messages, ensuring their integrity. To create a signature, an elliptic curve, generator point G, and range n must be specified, similar to the ECDH algorithm.

To calculate the signature, the signing party must have already generated a valid key-pair value. $privKey$ denotes the private key of the signing party. It is within the range of $< 1, n-1 >$. $pubKey$ denotes the corresponding public key generated as $privKey \cdot G$.

The signature is calculated as such:

1) A message hash, denoted as h, is computed using a cryptographic hash function.
2) A random number k is generated within the range of $< 1, n-1 >$.
3) A random point R is obtained by multiplying the generator point G by the generated random number k: $R = k \cdot G$. Only the x-axis value of the point is used and it's denoted as r.
4) Signature s of the hash is computed as
$$s = k^{-1} \cdot (h + r \cdot privKey) \equiv 1 (mod\ n)$$
where k^{-1} is a modular inverse.
5) The algorithm returns a completed signature in the form of a pair (r, s).

To verify the signature on the other end of the communication, following steps are taken:

1) The message hash h is calculated using the same cryptographic hash function that was used to generate the signature.
2) A modular inverse s_1 is calculated from s:
$$s_1 = s^{-1} (mod\ n)$$
3) The same random point R that was used during the signing process is calculated again:
$$R = (h \cdot s_1) \cdot G + (r \cdot s_1) \cdot pubKey$$
Only its x-axis value is used. This value will be denoted as r'.
4) Ensure that the generated value of r' matches the one provided in the signature: $r == r'$

Encryption – ECIES

The Elliptic Curve Integrated Encryption Scheme (ECIES) [10] is a scheme that provides a concept for implementing encryption using Elliptic Curves. The specific details of implementation are left to the programmer. ECIES generates its encryption key from the Elliptic Curve Diffie-Hellman (ECDH) algorithm.

III. Implementation issues

This section focuses on the usage of elliptic curves from a programmer's perspective, specifically the issues that arise when implementing elliptic curve cryptography. The goal is to present a selection of these issues.

A. Selecting the curve

Numerous elliptic curves are available, and selecting the most suitable one for a particular use case depends on various factors. For instance, when dealing with a smart card that has limited processing power and energy requirements, it is preferable to use a curve that requires fewer CPU cycles to compute a multiplication than one that requires more. Many of the properties of these curves can be compared.

A study [11] was made to compare some of the commonly used curves – *Curve25519* [12], *NIST P256* [13] and *Brainpool P256r1* [14].

The study compared curves and measured their performance across various cryptographic operations and parameters. The operations were implemented in several different ways, such as attempting to achieve the fastest implementation or the implementation requiring the lowest power. All of the curves underwent testing in the same environment. The main characteristics, speed (measured as CPU cycles, the fewer the better), and power consumption are shown in Figure 4.

As shown in the figure, Curve25519 appears to be the most efficient of the three options in terms of both required CPU cycles and total power consumption across all settings.

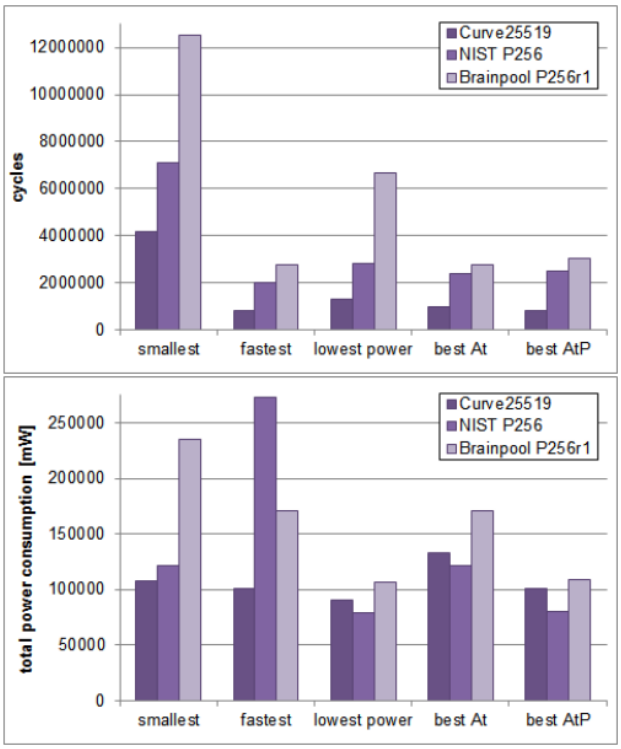

Fig. 4: Speed [in cpu-cycles] and power consumption comparison of different elliptic curves [11]

Algorithm 1 Double-And-Add Algorithm

Input: point P, n $\in \mathbb{N}$, k = number of bits in n
Output: n · P

$R \leftarrow \infty$ ▷ Point at infinity, represented as zero in practice.
▷ Move downwards across bit representation of n
for i **in range** $(k-1, 0, -1)$ **do**
  $R \leftarrow 2 \cdot R$
  if n_i == 1 **then**
    $R \leftarrow R + P$
  end if
end for
return Q

Algorithm 2 Double-And-Add-Always Algorithm

Input: point P, n $\in \mathbb{N}$, k = number of bits in n
Output: n · P

$R_0 \leftarrow \infty$
for i **in range** $(k-1, 0, -1)$ **do**
  $R_0 \leftarrow 2 \cdot R_0$
  $R_1 \leftarrow R_0 + P$
  $R_0 \leftarrow R_{k_i}$
end for
return Q_0

B. Selecting the multiplication algorithm

Multiplication was described in the section II-B. It cannot be implemented using repeated addition due to its low performance. The algorithms that use elliptic curves perform multiplication many times, making it the most cost-heavy operation due to its complexity. However, as will be presented, implementing a more effective can bring along security issues. Many different algorithms to increase the multiplication performance exist.

Double-And-Add

As was already briefly mentioned, a *double-and-add* [7], [15] algorithm is an often given example. It is similar to the *square-and-multiply* algorithm. It works by traversing the bits that make up the scalar component in a downward direction. Depending on whether the bit is 1 or 0, an operation is performed. Algorithm 1 shows how it works. A variation of this algorithm also exists, which traverses the bits from the lowest bit upwards and works in a similar manner.

Double-And-Add-Always

Although the Double-And-Add algorithm is faster than repeated addition for multiplication, it also presents a problem.

The algorithm is susceptible to a *side-channel* attack (which will be more explained later), which enables the attacker to detect when the addition is performed based, for example, on the energy requirements of the operation. This vulnerability could allow the attacker to determine the scalar component n, leading to security issues. [7], [15]

The Double-And-Add-Always [7], [15] algorithm was created to address the side-channel vulnerability of the Double-And-Add algorithm. This is achieved by performing an addition each time, making it difficult for attackers to determine when the bit is 1. However, this results in worse time complexity. The reason is that when the bit is 0, the calculated addition is not used but had to be computed anyway. This is illustrated in Algorithm 2.

Montgomery Ladder

The Double-And-Add algorithm and its vulnerability were presented, and the vulnerability was fixed by the Double-And-Add-Always algorithm. Same as the fixed version, Montgomery Ladder [7], [15] is resistant against side-channel attacks.

The Montgomery Ladder algorithm can be parallelized, leading to improved performance [15]. The algorithm can be further optimized, but the simplest version is shown in Algorithm 3.

C. Creating a public key

Generating a public key may be problematic [16]. That is due to issues with random number generation, particularly in embedded devices with low entropy. The public key is calculated by multiplying the private key with the generator point $privKey \cdot G$.

If a faulty random number generator was used, the $privKey$ random value may have already been used by another user.

According to the properties of elliptic curves, if two users share a private key (and the generator point G), they also share

Algorithm 3 Montgomery Ladder Algorithm

Input: point P, n $\in \mathbb{N}$, k = number of bits in n
Output: n $\cdot$ P

$R_0 \leftarrow \infty$
$R_1 \leftarrow P$
for i **in range** $(k-1, 0, -1)$ **do**
  if n_i == 0 **then**
    $R_1 \leftarrow R_0 + R_1$
    $R_0 \leftarrow 2 \cdot R_0$
  else
    $R_0 \leftarrow R_0 + R_1$
    $R_1 \leftarrow 2 \cdot R_1$
  end if
end for
return Q_0

a public key. This implies that if two users use the same faulty random number generator, they may obtain the same keys. In such a scenario, if an attacker discovers the private key of one user, they would not need to calculate the private key of the other person, but could simply use the key they already possess.

D. Validating given point

When performing calculations involving elliptic curves, such as calculating a new point when using ECDH, it is crucial to ensure that the given point is valid [17]. Failure to do so could enable an attacker to obtain information about the other party's private key. While some cryptography protocols may function without validation, it is essential to always validate the given point to ensure safety.

The conditions [17] the given point P comprised of coordinates (x, y) has to satisfy are listed below:

- P must not be a point at infinity: $P \neq \infty$. In practical terms, the point must not be zero.
- (x, y) belong to the field of integers of the curve: $0 \leq x, y \leq p-1$ where p is the modulo value the curve uses.
- P satisfies the defining curve equation
- $n \cdot P = \infty$ where n is the order of P.

All these conditions need to be satisfied for the given point to be valid.

IV. Attacks on Elliptic Curves

This section presents attacks on Elliptic Curves. These attacks are primarily based on existing implementation issues rather than on the mathematical foundations.

A. Brute-Force attack

The brute-force attack is the most basic method of attack. It involves trying every possible combination to find the private key from the public key, which is computed as $pubKey = privKey \cdot G$ where G is the generator point. This method works by attempting every possible value of the private key and matching it against the resulting public key. However, due to the length of the keys and the vast number of possible values for the private key, this type of attack is not feasible in real-world scenarios. It is impossible to defend against.

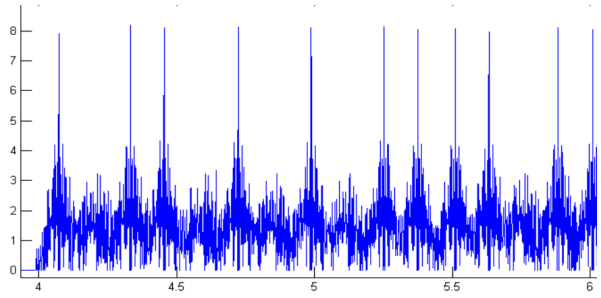

Fig. 5: Power consumption analysis used on Double-And-Add algorithm[4]

B. Side-channel attack

A side-channel attack involves monitoring a device for unintended information leaks, which may take the form of power consumption. While power consumption may not be an important statistic on its own, it becomes problematic in the context of certain algorithms, such as elliptic curve multiplication. The power consumption analysis side-channel attack can be split into two main subcategories. Those subcategories are *Simple Power Analysis* (SPA) and *Differential Power Analysis* (DPA). The main difference between them is that SPA can retrieve all necessary information to find the required value in a single power consumption observation of the algorithm run, while DPA usually requires multiple observations to find the required value. [18], [19]

SPA Attack

To demonstrate a SPA attack, we will utilize the previously mentioned Double-And-Add algorithm (see 1). As previously mentioned, power consumption leakage can be problematic, potentially enabling the attacker to determine the scalar component of the multiplication.

The Double-And-Add algorithm performs addition only when the current bit of the scalar component is 1. However, this can cause power spikes during loop iterations, which can be exploited by attackers to deduce each bit of the number. By combining this information, the attacker can deduce the entire scalar component, resulting in a serious security breach. [7]

This approach is illustrated in Figure 5. The correlation corresponds to the following sequence: Double and Add (DA = 1) was performed, followed by only a doubling (D = 0), followed by DA again... resulting in the first nine bits corresponding to $DA, D, DA, DA, D, DA, D, DA, D$.

A possible way to eliminate the information leakage was also previously shown in the Double-And-Add-Always algorithm. This algorithm performs the addition regardless of the current bit, making power spikes occur in each loop iteration and rendering the leaked information as a whole

[4]Image taken from https://cosade.telecom-paristech.fr/presentations/s2_p2.pdf

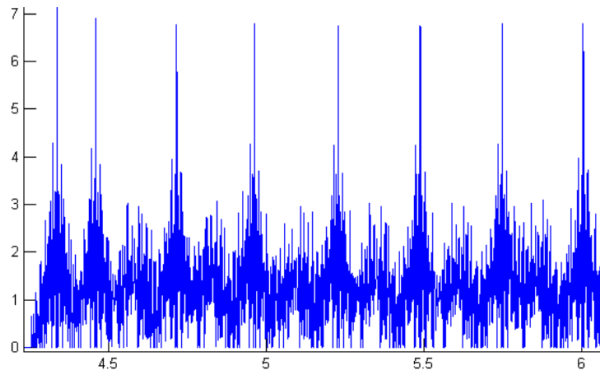

Fig. 6: Power consumption analysis used on Double-And-Add-Always algorithm[5]

useless. To defend against SPA side-channel attacks, it is necessary to interfere with the leaked information and make it indistinguishable from the rest of the program for the attacker. [7]

Figure 6 shows the power consumption of the Double-And-Add always algorithm. It can be observed that the power consumption remains the same in each iteration due to the dummy addition, with the spike at the beginning representing the initialization.

DPA Attack

To illustrate a DPA attack, we will use the zero-value attack [20]. Its goal is to determine the secret scalar component of the multiplication by using specific generator points. For this attack to be successful, the attacker must have the ability to select the generator point, and the scalar component must remain constant. This attack can bypass the Double-And-Add-Always algorithm shown in 2.

Through side-channel analysis, the attacker attempts to identify whether any of the registers used in computation contain a zero value. The success of the attack is heavily reliant on the implementation of the curve operations. It should be noted that this attack is only effective on certain curves that possess the necessary properties. If they do not have the required properties listed in [20], the attack can not be used.

DPA attacks can be resisted by randomizing the generator point or other values (eg. the scalar component of the point multiplication) which would otherwise be constant.

V. Conclusion

This work explores elliptic curves from a practical perspective, presenting the basics of the concept and its applications, without delving into sophisticated mathematics in the background.

The text presented several implementation issues, such as selecting the appropriate curve, preventing the attacker from accessing secret information through validated points, and choosing the correct multiplication algorithm and its relevant considerations.

Afterwards, attacks on elliptic curves exploiting the presented issues were shown.

One issue identified during the writing of this paper is the lack of direct comparisons among different elliptic curves. Comprehensive studies could be conducted to compare many of the known and commonly used elliptic curves.

Another issue is comprehending the mathematics behind elliptic curve cryptography. There are no readily accessible easy-to-understand resources that explain the necessary mathematics to someone without an advanced mathematical background.

However, even without a deep understanding of how elliptic curves work, a programmer can still create a secure implementation. This makes elliptic curves a popular method for solving cryptography problems.

[5]Image taken from https://cosade.telecom-paristech.fr/presentations/s2_p2.pdf

References

[1] A. Elngar, V. Balas, M. Kayed, and S. Panda, *Bitcoin and Blockchain. History and Current Applications*, 04 2020.

[2] M. Li, "The advance of ethereum digital signature," *Highlights in Science, Engineering and Technology*, vol. 39, pp. 1159–1163, 04 2023.

[3] V. Gupta, S. Gupta, and S. Shantz, "Performance analysis of elliptic curve cryptography for ssl," 09 2002.

[4] Y. Nir, S. Josefsson, and M. Pégourié-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier," RFC 8422, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8422

[5] P. Dzurenda, S. Ricci, J. Hajny, and L. Malina, "Performance analysis and comparison of different elliptic curves on smart cards," 08 2017.

[6] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to Elliptic Curve Cryptography*, 01 2004. [Online]. Available: https://link.springer.com/book/10.1007/b97644

[7] G. Štěpánka, "Algoritmy pro skalární násobení na eliptických křivkách a jejich využití v kryptografii [online]," 2019. [Online]. Available: https://is.muni.cz/th/rpvte/

[8] R. Haakegaard and J. Lang, "The elliptic curve diffie-hellman (ecdh)," 12 2015. [Online]. Available: http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf

[9] S. Nakov. Ecdsa: Elliptic curve signatures. [Online]. Available: https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages

[10] S. Nakov. Ecies hybrid encryption scheme. [Online]. Available: https://cryptobook.nakov.com/asymmetric-key-ciphers/ecies-public-key-encryption

[11] W. Wieser, "Comparison of different 256-bit elliptic curves for low-resource devices," Master's thesis, Graz University of Technology - Institute for Applied Information Processing and Communications, 2014. [Online]. Available: https://diglib.tugraz.at/download.php?id=576a7368c34ee&location=browse

[12] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228.

[13] Certicom Research, "Standards for Efficient Cryptography - SEC 2: Recommended Elliptic Curve Domain Parameters," Jan. 2010, section 2.4.2. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[14] J. Merkle and M. Lochter, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation," RFC 5639, Mar. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5639

[15] G. Gsenger, "Speeding up elliptic curve cryptography by optimizing scalar multiplication in software implementations," Master's thesis, Graz University of Technology - Institute for Applied Information Processing and Communications, 2014. [Online]. Available: https://diglib.tugraz.at/download.php?id=576a72d77c8c3&location=browse

[16] J. Bos, J. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," vol. 8437, 03 2014.

[17] A. Antipa, D. Brown, A. Menezes, R. Struik, and S. Vanstone, "Validation of elliptic curve public keys," in *Public Key Cryptography — PKC 2003*, Y. G. Desmedt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 211–223.

[18] E. K. Reddy, "Elliptic curve cryptosystems and side-channel attacks." *Int. J. Netw. Secur.*, vol. 12, no. 3, pp. 151–158, 2011.

[19] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Cryptographic Hardware and Embedded Systems*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 292–302.

[20] T. Akishita and T. Takagi, "Zero-value point attacks on elliptic curve cryptosystem," in *Information Security*, C. Boyd and W. Mao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 218–233.