Comparison of Digital Signature Algorithms based on Elliptic curves for Blockchain Technologies

Sergii Khrypko

Alfred Nobel University, Ukraine

ur9qq@ukr.net

Abstract

Blockchain is based on complex mathematical methods such as public key cryptography, hash functions, and digital signatures. They ensure the security and immutability of data. The question arises – which of the elliptic curve digital signature algorithms provides the optimal balance between speed, attack resistance, and compactness in the blockchain environment? The Schnorr algorithm demonstrates higher performance, reduced signature size, and improved support for multisignature schemes compared to traditional ECDSA, making it a promising choice for next-generation blockchain protocols. Therefore, a critical analysis of the efficiency and robustness of digital signature algorithms based on elliptic curves (ECDSA, EdDSA, Schnorr) in the context of blockchain technology is the task of this work, aimed at assessing their suitability for modern security and performance requirements.

Keywords: Ecdsa, Eddsa, Schnorr, Blockchain Technologies, Elliptic Curves

1. Introduction.

The task of this work is to critically analyze the efficiency and stability of digital signature algorithms based on elliptic curves (ECDSA, EdDSA, Schnorr) in the context of blockchain technologies in order to assess their compliance with modern security and performance requirements.

The mathematical methods that underlie blockchain are elliptic curves, ECDSA, and keys [1]. These principles are consistent with previously formalized mathematical methods for blockchain systems [2].

A fundamental part of blockchain is cryptographic algorithms. One such algorithm is ECDSA – Elliptic Curve Digital Signature Algorithm, which uses elliptic curves and finite fields for signing data. It is designed to allow a third party to verify the authenticity of the signature, eliminating the possibility of forgery. ECDSA signature and verification use different procedures consisting of several arithmetic operations.

Elliptic Curves.

Let K be a field. An elliptic curve over this field is a cubic curve defined by a cubic equation with coefficients from K and a point at infinity. One common form of elliptic curves is the Weierstrass form:

$$y^2 = x^3 + ax + b \tag{1}$$

In Bitcoin, the elliptic curve secp256k1 is used, where a = 0 and b = 7. Elliptic curves have several properties: for example, a non-vertical line that intersects two distinct points P and Q on the curve also intersects a third point R. Point addition and doubling are defined using these geometric properties, and scalar multiplication R = kP is defined by repeated addition.

2. Methodology.

The specific implementation used in Bitcoin is secp256k1, part of the Standards for Efficient Cryptography (SEC). In ECDSA, the private key is a random number between 1 and the order of the base point G. The public key is computed as:

$$\text{Public key} = \text{private key} \cdot G$$

This demonstrates that the maximum number of private keys (and therefore Bitcoin addresses) is normal, and equal to the order. However, the order is an incredibly large number, namely 2^{256} to the power, so it is unrealistic to accidentally or intentionally pick up another user's private key with modern technology. In a continuous field we can plot a tangent line and determine the public key on the graph, but there are some equations that do the same thing in the context of finite fields. The pointwise addition of p + q to find r is defined component-wise as follows:

$$c = (q\,y - p\,y) / (q\,x - p\,x), \quad \cdot r\,x = c\,2 - p\,x - q\,x, \quad r\,y = c\cdot(p\,x - r\,x) - p\,y \qquad (2)$$

The dot product of p to find r looks like this:

$$c = (3\cdot p\,x\,2 + a) / 2p\,y, \qquad r\,x = c\,2 - 2\cdot p\,x, r\,y = c\cdot(p\,x - r\,x) - p\,y \qquad (3)$$

In practice, the calculation of the public key is broken down into a series of point product and addition

operations, starting from a base point.

Let's look at an example from the other side, using small numbers, to get an idea of how keys are created and used for signing and verification.

To illustrate the principle, consider a toy example with small parameters:

- Equation: $y^2 = x^3 + 7$ (i.e. a = 0 and b = 7)
- Number by module: 67
- Base point: (2, 22)
- Order: 79
- Private key: 2

First, we will find the public key. Since we have chosen the simplest possible private key with a value of 2, this will only require the operation of doubling one point from the base point.
The calculation is presented as follows:

$$c = (3\cdot 22 + 0) / (2\cdot 22) \bmod 67c = (3\cdot 4) / (44) \bmod 67c = 12/44 \bmod 67$$

But, the result must be an integer. So, we need to multiply by the reciprocal, which space does not allow us

to specify here. In this case, we will have to trust ourselves for the moment that:

$$44\ {-1} = 32$$

Calculate:

$$c = 12\cdot 32 \bmod 67, \quad c = 49\ r\,x = (49\,2 - 2\cdot 2) \bmod 67r\,x = 52\ r\,y = (49\cdot(2 - 52) - 22) \bmod 67\ r\,y = 7.$$

The calculation of the public key is performed using the same operations of doubling and adding points. This is a trivial task that a regular personal computer or smartphone can solve in milliseconds. But the inverse problem (deriving the private key from the public key) is a discrete logarithm problem, which is considered and is currently computationally difficult. The best-known algorithms for solving it, such as Pollard's, have exponential complexity.

For secp256k1, to solve this problem, you need to do about 2 to the 256th power of operations, which requires computing time on a regular computer compared to the time of the existence of the universe. Therefore, the transition from a private key to a public key is by design a one-way trip.

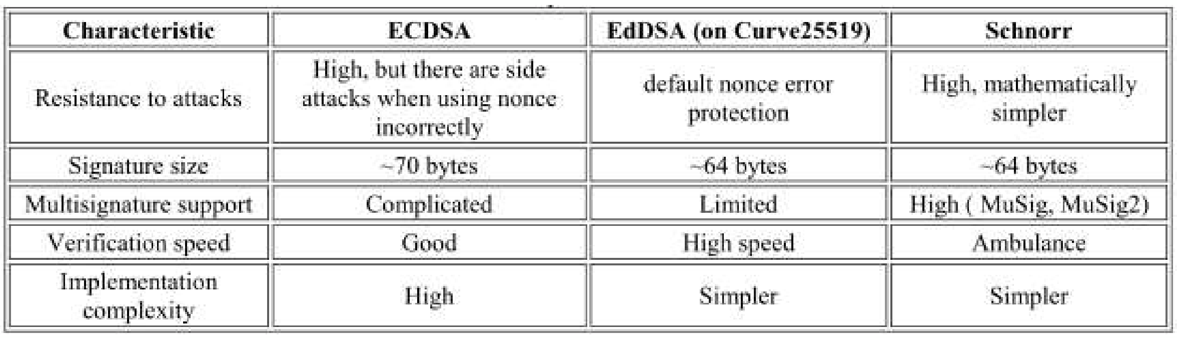
As with the private key, the public key is usually represented as a hexadecimal string. In an uncompressed public key, two 256-bit numbers representing the x and y coordinates are simply strung together into one long string. You can use the symmetry of the elliptic curve to create a compressed public key, storing only the x value and determining which half of the curve the point is on. From this partial information, you can reconstruct both coordinates.

Given the increasing demands for scalability, security, and efficiency of blockchain systems, special attention should be paid to a critical analysis of the digital signature algorithms underlying transactions. It is known that the ECDSA algorithm used in Bitcoin and many other blockchain platforms has a number of limitations, in particular in the area of protection against nonce misuse, the complexity of implementing multisignature, and the inefficient size of signatures.

3. Discussion.

The alternatives to ECDSA have significant advantages in terms of performance, security, and multi-user support. In particular: (Table 1):
– Schnorr is optimal for multisignature implementation and reducing blockchain data size.
– Ed25519 is a universal solution for fast signing and secure authentication.
Their adoption in real systems demonstrates the evolution of cryptographic standards towards greater efficiency, confidentiality, and scalability.

Table 1. Comparison of alternatives



Characteristic	ECDSA	EdDSA (on Curve25519)	Schnorr
Resistance to attacks	High, but there are side attacks when using nonce incorrectly	default nonce error protection	High, mathematically simpler
Signature size	~70 bytes	~64 bytes	~64 bytes
Multisignature support	Complicated	Limited	High (MuSig, MuSig2)
Verification speed	Good	High speed	Ambulance
Implementation complexity	High	Simpler	Simpler

The research objective was to evaluate the performance of the most common digital signature algorithms on elliptic curves in the context of blockchain applications: key generation time, signature creation and verification.

4. Results.

The experimental evaluation was conducted in Python 3.11 using libraries: ecdsa for ECDSA, PyNaCl for Ed25519, and coincurve for Schnorr (secp256k1). Testing was performed on Intel Core i5-8250U, 16GB RAM, Windows 11. Average execution times (1000 runs).

The total operation time for each algorithm was:

– ECDSA time : 0.003006458282470703

– EdDSA time : 0.003509044647216797

– Schnorr time : 0.0030078887939453125

The results are shown in the Table 2 and in Fig. 1.

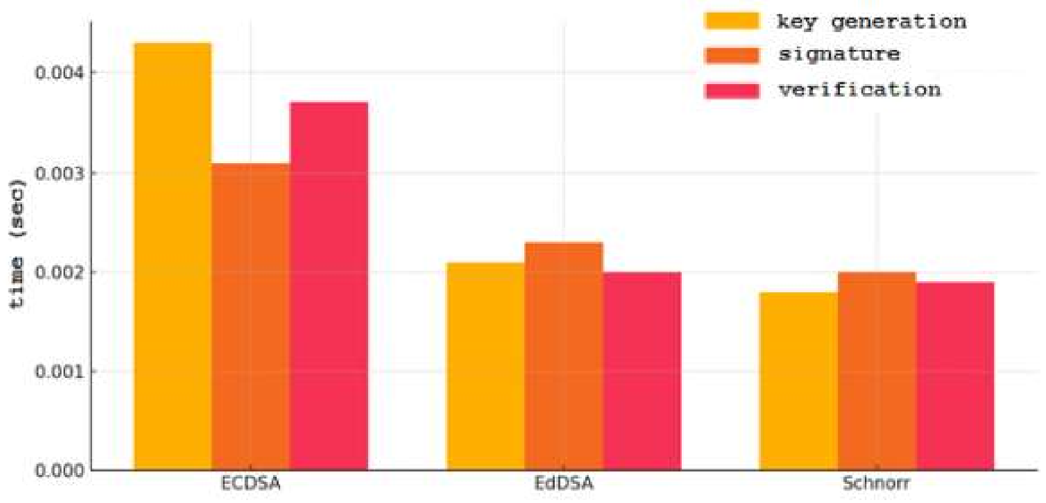


Fig.1. Summary of Timing Results (average of 1000 runs)

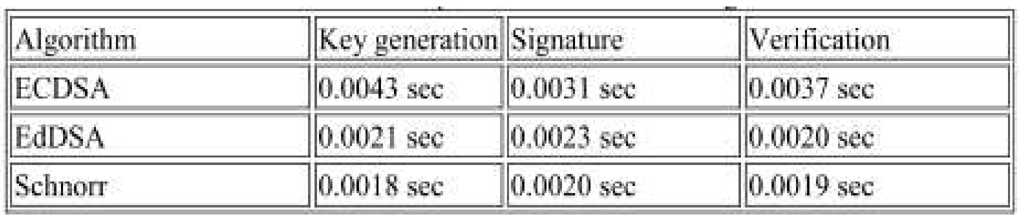
First, the keys and signatures were generated. Then the time for each algorithm was measured. Using algorithm functions were accepted generation keys, signature and verification. After which the execution time of each stage was measured.

The generalized value of time consumption was calculated for the function :

$$T\ sig\ (x) = a \cdot \log 2\ (n) + b$$

where x - key length, n - number of repetitions.

Table 2. Operation time for each algorithm



Algorithm	Key generation	Signature	Verification
ECDSA	0.0043 sec	0.0031 sec	0.0037 sec
EdDSA	0.0021 sec	0.0023 sec	0.0020 sec
Schnorr	0.0018 sec	0.0020 sec	0.0019 sec

To demonstrate the trade-off between the size of a cryptographic signature and the speed of its verification for different digital signature algorithms, a combined graph of the type was constructed. Therefore, the chart is useful for justifying algorithm selection in systems with limited computational or memory resources.

Schnorr and Ed25519 outperform ECDSA in both signature size and verification speed. Schnorr additionally supports multisignatures and aggregated verification, making it ideal for scalable blockchain systems. EdDSA provides strong resistance to nonce misuse attacks, making it reliable in high-load systems.

EdDSA algorithm (in particular, Ed25519) has also demonstrated high performance, especially in signature verification, as well as resistance to attacks related to the incorrect use of pseudo-random nonce values, making it a reliable alternative to ECDSA in high-load systems.

Instead, ECDSA, while remaining the most widely used algorithm in classical blockchain systems (Bitcoin, Ethereum), is inferior to modern alternatives in both performance and structural flexibility. The experimental results are consistent with the previously formulated hypothesis regarding the advantages of the Schnorr algorithm in the context of security, speed, and scalability. In addition, an example with a public key obtained for a private key 2 in a finite field (modulus 67), corresponding to the point (52, 7), confirms the correctness of the implementation of elliptic cryptography algorithms under simulation conditions. This fragment confirms the principles underlying the mathematical basis of the blockchain and demonstrates the application of basic operations on elliptic curves in practice.

Thus, the results obtained can serve as a basis for further research, and can also be used to select the optimal signature algorithm when designing new generation blockchain systems, in particular in systems that require high performance, support for multisignatures or data compression, and with the selection of optimal cryptographic algorithms for different types of blockchain protocols, considering their limitations and scalability requirements.

5. Conclusion.

This study provided theoretical analysis and experimental evaluation of digital signature algorithms based on elliptic curves (ECDSA, EdDSA, Schnorr) in blockchain applications. The Schnorr algorithm demonstrated the highest performance across all cryptographic operations, along with simplified multisignature schemes. The experimental results confirm its advantages in speed, security, and scalability. EdDSA also proved to be highly efficient, while ECDSA remains widely used but less flexible.

References:

1. Johnson D., Menezes A., Vanstone S. The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security, vol. 1, no. 1, pp. 36-63, 2001.
   URL: https://link.springer.com/article/10.1007/s102070100002
2. Sherbakov S., Khrypko S. Mathematical methods that underlie blockchain. Proceedings of the XI International Scientific and Practical Conference ISMSC-24, Zagreb, Croatia, Feb 21-23, 2024. pp.60-66.
   URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/94852/1/Brovkina_argot.pdf
3. Vaudenay S. The Security of DSA and ECDSA. International Workshop on Public Key Cryptography, Springer, Berlin, Heidelberg, pp. 309-323, 2003.
   URL: https://link.springer.com/content/pdf/10.1007/3-540-36288-6_23.pdf