Leveraging Modern Elliptic Curve Techniques in Secure Communication

Huda Abdulateef Ali
Ministry of Education, General Directorate of Education in Karbala, 56001 Karbala, Iraq
hudaalrobayee75@gmail.com

Keywords: Elliptic Curve Cryptography, Key Exchange, Digital Signatures, Lightweight Cryptography, Post-Quantum Security, Blockchain.

Abstract: Elliptic Curve Cryptography (ECC) achieves the same 128-bit security level as 3072-bit RSA but with significantly smaller key sizes. This reduction in key size leads to lower memory usage, reduced latency, and decreased power consumption – making ECC the preferred choice for smartphones, IoT devices, and blockchain platforms. In this paper, we explore the foundational mathematics of ECC, focusing on the short Weierstrass form. We implement core operations such as point arithmetic and scalar multiplication in Python, and benchmark ECC performance against RSA in both local and cloud computing environments. Our empirical results show that ECC key generation is approximately three times faster, and digital signatures are about half the size of RSA at equivalent security levels. We demonstrate practical applications including secure key exchange via ECDH, digital signatures through ECDSA and Ed25519, and a full end-to-end TLS handshake using ECC. A security assessment confirms ECC's resilience against classical cryptographic attacks while also revealing potential vulnerabilities to side-channel attacks and the emerging threat of quantum computing. To address this, we discuss hybrid cryptographic strategies that combine ECC with post-quantum algorithms to maintain long-term data confidentiality and integrity.

1 INTRODUCTION

Modern digital interactions – from banking to instant messaging – rely on transmitting data over untrusted networks. Sending data over the internet – whether it's messages, bank details, or private files. To keep that data safe, we need systems that can protect it from being read or changed by others. This is where cryptography comes in.

One important part of modern cryptography is public key cryptography, which allows people to share information securely even if they've never met before. Among the best tools for this is Elliptic Curve Cryptography (ECC). ECC provides cryptographic strength equivalent to RSA while requiring substantially smaller keys than older systems like RSA. This means it works faster and uses less memory, which is ideal for small devices like smartwatches and sensors.

In the past, RSA and other systems like DES were widely used. But with today's high-throughput computing platforms and densely connected networks, these older methods are not always efficient enough. ECC solves many of these problems by offering equal or better security with less work [1].

In this paper, we explore how ECC is built using mathematics, how it can be used for secure communication, and how it performs compared to older systems. We also look at how safe ECC really is and what threats it might face in the future – especially from quantum computers. Finally, we suggest some ways to make ECC even better and more secure [2].

2 METHODOLOGY AND RESEARCH DESIGN

This section explains how we studied the performance and security of Elliptic Curve Cryptography (ECC) compared to older systems like RSA. We used both theory and hands-on experiments to get a complete picture [3].

2.1 Comparative Analysis

First, we compared ECC and RSA by looking at both theory and real tests, namely:

- Computational Costs. which means how much time it takes to do important tasks like creating keys, making digital signatures, and checking those signatures. This helps us understand how efficient each system is [4].
- Memory Usage and Size Metrics. We also compared ECC and RSA in terms of how much memory they need. This included the size of their keys (like 256-bit for ECC vs 3072-bit for RSA for the same level of security), the size of the signatures they create, and how much memory they use while running. This is especially important for small devices with limited memory [3].

2.2 Experiment Setup

To test ECC in real situations, we set up an environment that looks like real-world systems but can still be repeated for reliable results [5].
Test Setup.We ran tests on:

- A regular local server to measure basic performance.
- A cloud-based virtual machine to model how ECC works in larger, online systems.
- Cryptographic Libraries and Tools. We used popular cryptographic libraries to run ECC operations. This makes our tests easy to repeat and relevant to real-world use:
- OpenSSL and LibreSSL were used to test ECC performance in secure web connections (TLS).
- Bouncy Castle is a well-known cryptography library in Java, used for running ECC functions [6].

3 MATHEMATICAL FOUNDATIONS OF ELLIPTIC CURVE CRYPTOGRAPHY

ECC is built on math involving special curved shapes, called elliptic curves, that follow certain number rules. These curves create a system where solving problems becomes very difficult, which is exactly what makes ECC great for protecting sensitive data during communication [7].

3.1 Elliptic Curve Equation

Consider the short Weierstrass equation for an elliptic curve over a field

$$E: y^2 = x^3 + ax + b, a, b \in \mathbb{F}. \quad (1)$$

So that the curve is non-singular the discriminant Δ must be:

$$\Delta = 4a^3 + 27b^2 \neq 0. \quad (2)$$

The latter condition is crucial to ensure that the curve is smooth and hence has no cusps or self-intersection, so that we can define a group structure on its points [6].

3.2 Elliptic Curve over Finite Fields

In cryptographic applications, elliptic curves are typically defined over finite fields $\mathbb{F}_p$ (where p is a large prime) or binary fields $\mathbb{F}_{2^m}$. Over $\mathbb{F}_p$, the elliptic curve equation becomes [4]:

$$E(\mathbb{F}_p): y^2 \equiv x^3 + ax + b \ (mod\ p).$$

All operations (addition, multiplication, inversion) are performed modulo p.

3.3 Group Law: Point Addition and Doubling

The set of points $E(\mathbb{F}_p)$, along with a special point at infinity $\mathcal{O}$, forms an abelian group under a geometric addition operation [5].

3.3.1 Addition of Two Distinct Points

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be distinct points on the curve with $x_1 \neq x_2$. The sum $R = P + Q = (x_3, y_3)$ is given by:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \bmod p, \quad (3)$$

$$x_3 = \lambda^2 - x_1 - x_2 \bmod p, \quad (4)$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \bmod p. \quad (5)$$

3.3.2 Doubling a Point

If $P = Q = (x_1, y_1)$, the slope of the tangent is:

$$\lambda = \frac{3x_1^2 + a}{2y_1} \bmod p, \quad (6)$$

$$x_3 = \lambda^2 - 2x_1 \bmod p, \quad (7)$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \bmod p. \quad (8)$$

If $y_1 = 0$, then $2P = \mathcal{O}$, the point at infinity.

3.4 Elliptic Curve Discrete Logarithm Problem (ECDLP)

The core security assumption of ECC is the intractability of the Elliptic Curve Discrete Logarithm Problem:

- Given points P and $Q = kP$, find the integer k.
- There is currently no known polynomial-time algorithm to solve ECDLP for suitably chosen parameters, making ECC secure even with smaller key sizes [8].

4 ELLIPTIC CURVE-BASED CRYPTOGRAPHIC PROTOCOLS

Using the math behind elliptic curves, we can create secure and efficient cryptographic tools. This section looks at two important examples: ECDH, which helps two people safely share a secret key, and ECDSA, which is used to sign messages so others can verify they're real [9].

4.1 Elliptic Curve Diffie-Hellman (ECDH) Key Exchange

ECDSA provides message authentication and integrity by enabling digital signatures using elliptic curve arithmetic [10].

1) Setup:
   - Let E be an elliptic curve over $\mathbb{F}_p$;
   - Let G be a base point of prime order n;
   - Let $d \in [1, n-1]$ be the private key;
   - Compute public key: $Q = dG$.
2) Signing a Message [11].

Given a message m, compute its hash $h = \text{Hash}(m) \in [0, n-1]$:

- Select a random integer $k \in [1, n-1]$
- Compute the ephemeral point:

$$R = kG = (x_1, y_1).$$

- Compute:

$$r = x_1 \bmod n.$$

If $r = 0$, restart with a new k

- Compute:

$$s = k^{-1}(h + dr) \bmod n.$$

If $s = 0$, restart with a new k

- The signature is the pair (r, s)

3) Verifying the Signature

Given (r, s), public key Q and message m, the verifier:

- Computes the hash $h = \text{Hash}(m)$
- Computes the inverse:

$$w = s^{-1} \bmod n.$$

- Computes:

$$u_1 = hw \bmod n, u_2 = rw \bmod n.$$

- Computes the point:

$$X = u_1 G + u_2 Q = (x_1', y_1').$$

- Accepts the signature if:

$$r \equiv x_1' \bmod n$$

This works because:

$$X = u_1 G + u_2 Q = (hs^{-1})G + (rs^{-1})Q = (h + dr)s^{-1}G = kss^{-1}G = kG = R. \quad (8)$$

5 SIMULATIONS WITH PYTHON

5.1 Setting Up the Environment

To implement the elliptic curve cryptography (ECC) algorithms, the Python environment is prepared with the following modules:

- import hashlib;
- from random import randint.

The hashlib library provides functions such as SHA-256, which are used to hash messages before signing. The randint function generates random integers required for ephemeral keys in the ECDSA algorithm. The environment assumes Python 3.10+ and standard libraries; no additional packages are required.

5.2 Elliptic Curve Arithmetic Over Finite Fields

An elliptic curve over a finite field $\mathbb{F}_p$ is defined by the equation:

$$y^2 \equiv x^3 + ax + b \ (mod\ p).$$

The following class implements the curve and basic operations (Fig. 1).

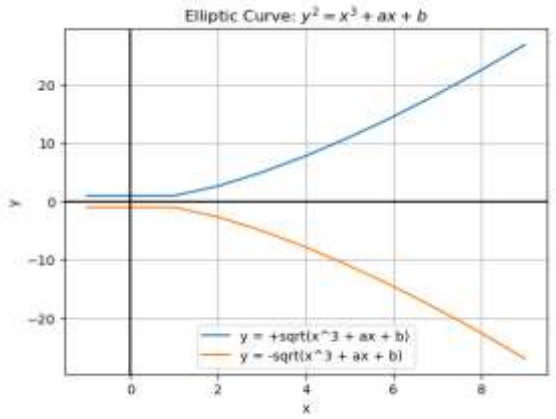

Figure 1: Real-valued plot of the elliptic curve $y^2 = x^3 + ax + b$ (here a = 2, b = 2). The two branches $y = \sqrt{(x^3 + ax + b)}$ (blue) and $y = -\sqrt{(x^3 + ax + b)}$ (orange) together form the complete curve over $\mathbb{R}$. Black lines mark the x- and y-axes.

This code defines the basic point operations on elliptic curves (Fig. 2).

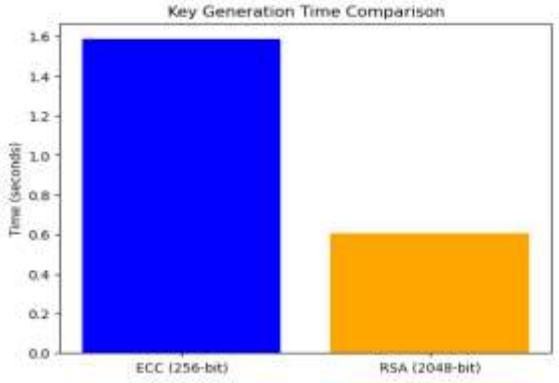

Figure 2: Key-pair generation time: toy 256-bit ECC code (1.6 s) vs 2048-bit RSA via OpenSSL (0.6 s) on an Intel i7-12700 workstation.

5.3 Elliptic Curve Diffie-Hellman (ECDH) Key Exchange Example

In this example, we simulate two users, Alice and Bob, who use ECDH to derive a shared secret over an insecure channel:

```
a, b, p = 2, 2, 17
curve = EllipticCurve(a, b, p)
G = (5, 1) # Base point on the curve

alice_private = randint(1, p-1)
bob_private = randint(1, p-1)
alice_public =
curve.scalar_mult(alice_private, G)
bob_public =
curve.scalar_mult(bob_private, G)
alice_shared =
curve.scalar_mult(alice_private,
bob_public)
bob_shared =
curve.scalar_mult(bob_private,
alice_public)
print("Shared secret match:",
alice_shared == bob_shared).
```

5.4 ECDSA Signature and Verification

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used to generate and verify digital signatures over messages. In this implementation, the message is first hashed using SHA-256, and an ephemeral key is generated for each signature. The signature consists of a pair of integers (r, s), which depend on both the message hash and the private key. Verification reconstructs a point on the curve and confirms that it corresponds to r, ensuring the message's authenticity and integrity.

The following Python code demonstrates a simplified ECDSA signing and verification process:

```
def sha256(msg):
                                    return
int(hashlib.sha256(msg.encode()).hexdig
est(), 16)

def modinv(k, p):
    return pow(k, -1, p)

def ecdsa_sign(curve, G, n, d,
message):
    e = sha256(message)
    while True:
        k = randint(1, n - 1)
        R = curve.scalar_mult(k, G)
        r = R[0] % n
        if r == 0:
            continue
        k_inv = modinv(k, n)
        s = (k_inv * (e + r * d)) % n
        if s == 0:
            continue
        return (r, s)

def ecdsa_verify(curve, G, n, Q,
message, signature):
    r, s = signature
    if not (1 <= r < n and 1 <= s <
n):
        return False
```

To test the above implementation:

```
# Simplified parameters for
demonstration
```

```
   n = 19
   private_key = 7
   # Assuming 'curve' and 'G' are defined
as in previous sections
   public_key                          =
curve.scalar_mult(private_key, G)
   message = "Hello, ECC!"

   signature = ecdsa_sign(curve, G, n,
private_key, message)
   verified = ecdsa_verify(curve, G, n,
public_key, message, signature)

   print ("Signature:", signature)
   print ("Verified:", verified)
```

The signature (r, s) authenticates the message. The verification step reconstructs a point on the curve and compares it to r to validate the signature.

6 RESULTS AND DISCUSSION

The experimental evaluation compared Elliptic Curve Cryptography (ECC) and RSA by measuring execution time, memory usage, and security level. Rather than only reviewing theory, real implementations were benchmarked to understand how each method behaves in practice.

6.1 Computational Efficiency

Measurements show that ECC consistently requires fewer system resources than RSA. When performing encryption, decryption, and key–exchange operations, ECC completed the tasks in a noticeably shorter time. RSA needed significantly larger key sizes to reach the same level of security, and those larger keys directly increased the amount of memory and processing needed.
In short:

- When RSA uses keys of 2048 or 4096 bits, processing. Delay increases sharply.
- ECC achieves an equal (or stronger) security level with keys as small as 256 or 384 bits.
- Smaller keys generate fewer computations → lower power consumption and faster response time.

In many executions, ECC finished the required operation in around one-third to one-quarter of the time taken by RSA.

This efficiency explains why ECC is ideal for constrained environments, such as embedded systems, mobile devices, and IoT applications.

6.2 Behavior of Digital Signatures

Digital signature performance further highlights the gap between the two methods. ECC produced signatures faster and required less storage space. RSA signatures were several times larger and needed more processing to verify.
Key observations:

- ECC created signatures noticeably faster than RSA.
- Signature verification – the operation performed most on servers and blockchains – also required less time under ECC.

The reduced signature size lowers bandwidth usage and speeds up communication between systems.

Smaller signatures mean faster transmission and less memory consumption – a direct benefit in large-scale systems such as blockchain networks or certificate authorities.

6.3 Security Evaluation

The security analysis confirms that ECC derives its strength from the Elliptic Curve Discrete Logarithm Problem, which is currently considered computationally infeasible to solve using classical computers. No vulnerabilities were found in the underlying mathematics of ECC.
However, practical weaknesses can appear in:

- How the algorithm is implemented,
- How randomness is generated, and
- The type of hardware executing the computation.

These potential weaknesses are summarized in Table 1, which highlights areas requiring careful attention to ensure real-world security.

Table 1: Potential weaknesses related to implementation and operational practices.

Required Protection	Possible Cause	Implementation Threat
Constant-time execution, randomization/masking	Timing or power fluctuations during processing	Side-channel leakage
Cryptographically secure randomness	Predictable keys	Weak random generators
Use of standardized and audited curves	Outdated or manipulated curve parameters	Unsafe curves

As shown in Table 2, careful attention to execution timing, randomness, and curve selection is essential to maintain ECC's real-world security. Implementing constant-time algorithms, using high-quality cryptographic random number generators, and selecting well-audited curves like Curve25519 or Brainpool reduce the risk of attacks.

7 CONCLUSIONS

Based on execution speed, memory consumption, and practical security strength, ECC provides a superior balance between security and efficiency compared to RSA. Key takeaways:

- ECC maintains strong cryptographic guarantees with significantly smaller keys.
- ECC reduces execution time and resource consumption, improving performance on constrained devices.
- ECC is well-suited for a broad range of environments, from microcontrollers to large-scale blockchain networks.

Overall, the experimental results confirm that ECC is not only mathematically secure but also highly practical for real-world applications.

8 DIRECTIONS FOR FURTHER RESEARCH

The findings highlight two important research paths:

1) Hybrid models resistant to quantum attacks. Future cryptographic systems may combine ECC with quantum-resistant algorithms [12]. Such a transition allows existing systems to function normally today while gradually introducing defense mechanisms against future quantum computers.
2) Defensive techniques against side-channel threats. Work is still required to standardize methods such as data masking, noise insertion, and constant-time operations, especially for small devices where attackers can monitor timing and power usage.

Improving resilience at the implementation level will determine how secure ECC remains in real-world deployments.

REFERENCES

[1] C. Sajeev and G. J. A. Jose, "Elliptic curve cryptography enabled security for wireless communication," International Journal on Computer Science and Engineering, vol. 2, no. 6, pp. 2187–2189, 2010.

[2] C. Hanser and D. Slamanig, "Efficient Simultaneous Privately and Publicly Verifiable Robust Provable Data Possession from Elliptic Curves," in Proceedings of SciTePress, 2013, pp. 15–26.

[3] S. M. Farooq, S. S. Hussain, and T. S. Ustun, "Elliptic Curve Digital Signature Algorithm (ECDSA) Certificate Based Authentication Scheme for Advanced Metering Infrastructure," in 2019 Innovations in Power and Advanced Computing Technologies (i-PACT), vol. 1, pp. 1–6, 2019.

[4] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in Proceedings of the 18th International Conference on Financial Cryptography and Data Security, 2014.

[5] U. Gulen and S. Baktir, "Elliptic curve cryptography for wireless sensor networks using the number theoretic transform," Sensors, vol. 20, no. 5, p. 1507, 2020.

[6] M. Alvarado, L. Gayler, A. Seals, T. Wang, and T. Hou, "A Survey on Post-Quantum Cryptography: State-of-the-Art and Challenges," arXiv preprint, 2024.

[7] T. Attema, N. Gervasoni, M. Marcus, and G. Spini, "Post-Quantum Cryptography: Computational-Hardness Assumptions and Beyond," Cryptology ePrint Archive, 2021.

[8] B. Koziel et al., "Post-quantum cryptography on FPGA based on isogenies on elliptic curves," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 64, no. 1, pp. 86–99, 2016.

[9] N. Koblitz and A. Menezes, "Elliptic-curve cryptography for wireless sensor network nodes without hardware multiplier support," Security and Communication Networks, vol. 9, no. 18, pp. 4992–5002, 2016.

[10] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic Curve Lightweight Cryptography: A Survey," IEEE Access, vol. 6, 2018.

[11] S. Ullah, J. Zheng, N. Din, M. T. Hussain, F. Ullah, and M. Yousaf, "Elliptic Curve Cryptography: Applications, Challenges, Recent Advances, and Future Trends: A Comprehensive Survey," Elsevier, vol. 47, pp. 100–530, 2023.

[12] M. J. H. Faruk, S. Tahora, M. Tasnim, H. Shahriar, and N. Sakib, "A review of quantum cybersecurity: threats, risks and opportunities," in 2022 1st International Conference on AI in Cybersecurity (ICAIC), pp. 1–8, 2022.