A comparative overview of cryptographic algorithms: RSA, ECC, AES, and ChaCha20

1st Luca Arborio	2nd Damine Bantos-Arnaud	3rd Paul Morel	4th Haseeb Ahmad	5th Sibghat Ullah
136749	136716	136763	135456	136364
University of Milan	*CYTECH*	*ISEP*	*ISCTE*	*ISCTE*
Milan, Italy	Cergy, France	Paris, France	Lisbon, Portugal	Lisbon, Portugal

***Abstract*—This work conducts a comparison between the four most popular cryptoalgorithm. Such as RSA, ECC, AES and Chacha20. It summarises the objectives of modern cryptography and differentiates between symmetric and asymmetric methods. Only a short structure, security assumption and common usage is described for each algorithm. The document also describes the measurement methodology that was used to assess the performance, which includes key generation, key exchange, encryption and decryption. With a theoretical background and well-defined experimental methodology, the project summarizes how these algorithms vary in performance, security and applicability to practical systems.**

***Index Terms*—Cryptography, Public-Key Cryptography, Symmetric Cryptography, RSA, ECC, AES, ChaCha20, Block Cipher, Stream Cipher, Integer Factorization, Modular Exponentiation, Scalar Multiplication, Key Exchange, Digital Signatures, Performance Analysis**

I. Introduction

Cryptography is an important part of information security, as it allows data to be protected from unauthorised access, alteration and forgery. It leverages mathematical techniques to mathematically process readable data (plaintext) and transformed it into gibberish (ciphertext) that can only be read by authorized parties. In modern times, cryptography provides secure communication, authentication, and data integrity for network systems. Confidentiality, authentication, integrity, and non-repudiation are four main objectives of cryptographic mechanisms. The implementations is on Github: https://github.com/Luke-MT/Network-and-Information-Systems-Security-Project

A. Main Goals of Cryptography

- **Confidentiality:** ensures that data is only accessible to authorized persons, when transmitted or stored. Images or touch-sensitive circuitry can also be encrypted using secret keys, and therefore made unreadable if they are eavesdropped upon.
- **Integrity:** Protects data from unauthorized modification. Hash functions and message authentication codes (MACs) are used to ensure that data hasn't been altered during transmission or storage..
- **Authentication:** Confirms the identity of communicating entities. Digital certificates and signatures verify that data originates from a trusted source.
- **Non-Repudiation:** ensures that the sender cannot deny having sent the message or performed an ction. They do so through the use of digital signatures, which offer irrefutable evidence of who issued a particular transaction.

B. Symmetric vs. Asymmetric Encryption

The table below compares the two major types of encryption.

TABLE I
Comparison of Symmetric and Asymmetric Encryption

Feature	Symmetric Encryption	Asymmetric Encryption
Key Structure	Uses a single shared secret key for encryption as well as decryption.	makes use of two mathematically related keys – a public key as well as a private key.
Examples	AES, ChaCha20	RSA, ECC
Speed	Faster and suitable for bulk data encryption.	Slower because of complex mathematical operations.
Security Basis	Security depends on keeping the shared key secret.	Security relies on complex mathematical problems (factoring, ECDLP).
Use Cases	File encryption, VPNs, data storage.	Digital signatures, secure key exchange, authentication.

In this work, symmetric encryption is AES and asymmetric methods are RSA and ECC

II. RSA: Rivest–Shamir–Adleman

That algorithm, known as RSA, after its inventors Ronald Rivest, Adi Shamir and Leonard Adleman was first introduced in 1977. The security is based on the assumed hardness of factoring a product of two large primes. (n=pq). Even though there are more efficient public-key primitives, RSA continues to remain in use; it is used even when other key exchange algorithms are recommended for performance. This paper abstracts RSA from the point of performance in theory as part

of a wider comparison in general with AES, ChaCha20 and ECC.

A. Mathematical Foundations

Let p and q be the two distinct random primes and $n = pq$. Define Euler's totient $\phi(n) = (p-1)(q-1)$. Choose a public exponent e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The private exponent d satisfies

$$d \equiv e^{-1} \pmod{\phi(n)}. \tag{1}$$

The public key is (e, n); the private key is (d, n). Encryption of $M \in \mathbb{Z}_n$ yields

$$C = M^e \bmod n, \tag{2}$$

and decryption is

$$M = C^d \bmod n. \tag{3}$$

Correctness follows from Euler's theorem since $M^{ed} \equiv M \pmod{n}$. Practical deployments use padding (e.g., OAEP) and standardized encoding as specified in PKCS.

B. Theoretical Performance

RSA's core operation is modular exponentiation on large integers. Using fast exponentiation and Montgomery reduction, the dominant cost scales superlinearly with the modulus bit-length k. In practice:

- **Encryption** is typically faster than decryption because implementations choose a small public exponent, commonly $e = 65537$.
- **Decryption** is heavier due to the large private exponent d. Chinese Remainder Theorem (CRT) optimizations reduce decryption time by roughly a factor of four by working modulo p and q.
- **Key generation** is computationally intensive: it requires generating strong random primes and running probabilistic primality tests.

C. Applications

RSA underpins key functionality in many protocols:

- **Key establishment in TLS/SSL:** RSA key transport or certificate-based authentication.
- **Digital signatures:** RSA-PSS is a modern, provably secure signature variant.
- **Secure email and software distribution:** PGP/S/MIME, package signing.

In most systems, RSA is used to protect a symmetric session key (e.g., for AES), which then encrypts bulk data efficiently.

D. Limitations and Trends

RSA's large key sizes and slow private-key operations make it suboptimal for constrained devices. Implementations must be hardened against side-channel attacks (timing, power analysis). At comparable security levels, ECC achieves far smaller keys and lower computational cost. Looking ahead, large-scale quantum computers running Shor's algorithm would break RSA; consequently, standards bodies are transitioning to post-quantum schemes for long-term security.

III. ECC: Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a modern type of public-key cryptography (PKC) based on elliptic curves' algebraic structure over finite fields. In the middle of the 1980s, Neal Koblitz and Victor Miller independently proposed ECC, offering a high degree of security with considerably**smaller key sizes** in contrast to more traditional public-key schemes like RSA. Because of its effectiveness, ECC is perfect for settings with limited resources, like mobile devices and the Internet of Things (IoT). ECC's security is based on the premise that the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**. Given a base point P on an elliptic curve and a resulting point Q (where $Q = kP$), it is computationally infeasible to determine the scalar integer k, which serves as the secret key.

A. Core Mathematical Concepts

The fundamental operation in ECC is **scalar multiplication**, denoted as $k \cdot P$. This is not a simple multiplication but rather the process of adding a point P to itself k times. This operation, called **point addition**, has a clear geometric definition. As shown in Fig. 1, to include two separate points P and Q, A straight line is drawn through them. At a third point, this line will cross the curve. R. The result of the addition $P + Q$ is the reflection of this point across the x-axis, which gives the point R. Scalar multiplication is simply the extension of this principle.

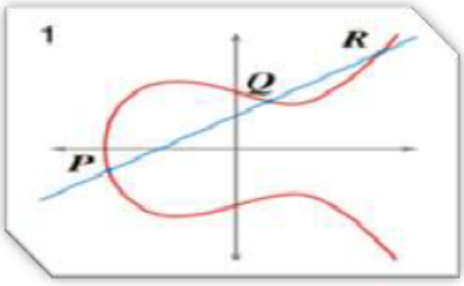

Fig. 1. Geometric representation of point addition ($P+Q = R$) on an elliptic curve.

B. Key Exchange with the ECDH Protocol

In ECC, the processes of "encryption" and "decryption" are most commonly realized through key agreement protocols like **Elliptic Curve Diffie-Hellman (ECDH)**,utilized to create a shared secret key between two people. The data is then encrypted using a symmetric cypher and this shared secret.

1) Public-Private Key Pair Generation: Alice and Bob first agree on public domain parameters: a specific elliptic curve E and a fixed **base point** G on that curve.

1) **Alice's Action:**
   - Alice selects a random, secret integer d_A as her **private key** ($d_A < n$).

- Alice computes her **public key** P_A by performing scalar multiplication on the base point G:

$$P_A = d_A \cdot G \tag{4}$$

2) **Bob's Action:**
   - Bob selects a random, secret integer d_B as his **private key** ($d_B < n$).
   - Bob computes his **public key** P_B:

$$P_B = d_B \cdot G \tag{5}$$

C. Shared Secret Calculation

To establish the shared secret, both parties perform a final scalar multiplication using their own private key and the other party's public key.

1) **Alice's Calculation:** Alice uses her private key d_A and Bob's public key P_B:

$$K = d_A \cdot P_B = d_A \cdot (d_B \cdot G) \tag{6}$$

2) **Bob's Calculation:** Bob uses his private key d_B and Alice's public key P_A:

$$K = d_B \cdot P_A = d_B \cdot (d_A \cdot G) \tag{7}$$

Due to the associative property of scalar multiplication, both parties arrive at the identical shared secret point $K = (K_x, K_y)$. The security of this key establishment relies on the fact that an eavesdropper, knowing only the public keys P_A, P_B and the public point G, cannot solve the ECDLP to find either of the private keys, d_A or d_B. The calculated point K is the shared secret. In practice, the parties do not use the point itself, but derive a key from it. Typically, the x-coordinate of K, K_x, is passed through a key that is appropriate for a symmetric encryption algorithm using a key derivation function (KDF).

D. Advantages and Practical Applications

ECC's ability to deliver comparable security with substantially smaller key sizes is its main advantage over its predecessors, resulting in notable performance gains.

1) Efficiency and Key Size:

- **Faster Computations:** Key generation and signing operations are significantly quicker.
- **Lower Power Consumption:** appropriate for battery-operated gadgets such as IoT sensors and smartphones.
- **Reduced Bandwidth and Storage:** Smaller keys and signatures require less space and data transfer.

2) Real-World Use Cases: ECC is utilized in many different applications and forms the basis of modern day digital security:

- **Web Security:** The Transport Layer Security (TLS) protocol protects HTTPS connections. Generally utilizes ECDH (more especially, ECDHE) as its key exchange mechanism.
- **Cryptocurrencies:** Bitcoin, Ethereum, and many other The Elliptic Curve Digital Signature Algorithm (ECDSA) is used by cryptocurrencies to secure wallets and approve transactions.
- **Secure Messaging:** End-to-end encrypted messaging apps like Signal and WhatsApp use the Signal Protocol, which relies on ECDH to establish secure communication channels between users.

IV. AES: The Advanced Encryption Standard

The **Advanced Encryption Standard (AES)**is a block cipher with symmetric keys that was standardized in 2001 by the National Institute of Standards and Technology (NIST) in the United States. The Belgian cryptographers Joan Daemen and Vincent Rijmen created it. the algorithm, originally named **Rijndael**, was selected through an open, multi-year competitionto take the place of the outdated Data Encryption Standard (DES). In contrast to asymmetric cryptosystems (like ECC or RSA) that use different keys for encryption and decryption, AES is a **symmetric** algorithm,indicating that both procedures use the same secret key. It works with fixed-size data blocks, specifically 128-bit blocks, and exists in three primary sizes:

- **AES-128:** 128-bit key (10 rounds of transformation)
- **AES-192:** 192-bit key (12 rounds of transformation)
- **AES-256:** 256-bit key (14 rounds of transformation)

Its combination of strong security, high performance, and implementation flexibility has made it the global standard for bulk data encryption.

A. The Inner Workings of AES

AES organizes a 128-bit block of plaintext into a 4x4 byte matrix known as the **State**. Then, over several rounds, the algorithm iteratively applies a number of mathematical transformations to this State. Confusion and diffusion, two fundamental concepts of contemporary cryptography that obfuscate the connection between the plaintext, ciphertext, and key, are intended to be introduced in each round.

A standard round in AES consists of four distinct transformation layers, each with a specific purpose.

1) The Four Transformation Layers:

1) **`SubBytes` (Substitution):** In this non-linear byte substitution step, each State byte is swapped out for a different one using a pre-established lookup table called the **Rijndael S-box**. This layer is critical for introducing confusion and is the primary source of AES's defense against linear and differential cryptanalysis.
2) **`ShiftRows` (Permutation):** The bytes in each State row are cyclically moved to the left in this step. One byte is shifted in the second row, but not in the first. the third by two, and the fourth by three. This transformation ensures that data from one column is spread across multiple columns in the next round, providing inter-column diffusion.
3) **`MixColumns` (Mixing):** This layer uses a particular mathematical operation over a finite field to combine the four bytes in each of the State's individual columns. (Galois Field $GF(2^8)$). The `MixColumns` step, along

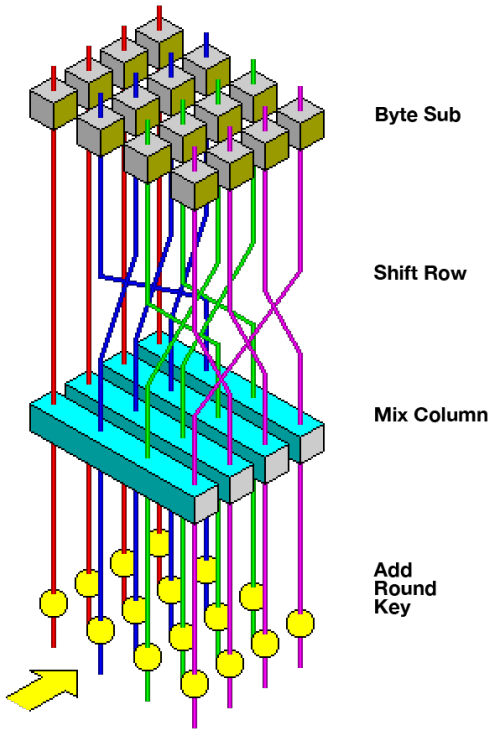

Fig. 2. Overview of the AES Encryption Process

with `ShiftRows`, is a key component of AES's diffusion mechanism, ensuring that a change in a single plaintext bit affects many ciphertext bits.

4) **`AddRoundKey` (Key Addition):**The State and a portion of the expanded key are combined in this round's final step. known as the **round key**. This is achieved through a simple bitwise XOR operation. This is the only step that directly involves the secret key, making the transformation dependent on it.

2) Key Expansion: Before the encryption process begins, the initial secret key (128, 192, or 256 bits) is expanded into a larger set of round keys using the **AES key schedule**. A unique round key is generated for the initial `AddRoundKey` step and for each of the subsequent rounds.

B. The Encryption and Decryption Process

The full AES encryption process for a single block of data follows these steps:

1) **Initial Key Addition:** The first round key is XORed with the plaintext state.
2) **Main Rounds:** The State undergoes $N - 1$ rounds of transformations, where N is the total number of rounds (10, 12, or 14). Each round consists of all four layers: `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`.
3) **Final Round:** A final, slightly modified round is performed, which includes `SubBytes`, `ShiftRows`, and `AddRoundKey`, but **omits** the `MixColumns` step.

The final State is the 128-bit ciphertext block.

Decryption is not a completely new algorithm but rather the reverse of the encryption process. It involves applying the inverse of each transformation layer (`InvSubBytes`, `InvShiftRows`, `InvMixColumns`) in the reverse order, using the same expanded set of round keys.

C. Security and Real-World Applications

1) Security and Resilience: AES is considered extremely secure. To date, no practical cryptographic attacks against AES exist. The only successful attacks are theoretical or side-channel attacks that exploit flawed implementations rather than the algorithm itself. The primary defense against a determined adversary is a **brute-force attack** trying every possible key. With key sizes of 128, 192, and 256 bits, the number of possible keys is so vast that such an attack is computationally infeasible with current and foreseeable technology.

2) Widespread Use Cases: Due to its security and performance, AES has been adopted as the encryption standard for a vast range of technologies:

- **Network Security:** It is a core component of protocols like TLS (securing HTTPS web traffic) and is used to encrypt data in Wi-Fi networks (WPA2 and WPA3).
- **File and Disk Encryption:** Tools like BitLocker (Windows), FileVault (macOS), and VeraCrypt use AES to protect data at rest.
- **Secure Messaging:** While protocols like ECDH are used for key exchange, AES is typically used for the actual encryption of messages in applications like Signal and WhatsApp.
- **Government and Military:** AES is approved by the U.S. government for protecting classified information, including Top Secret data when using the 256-bit key variant.

V. ChaCha20: A High-Speed Stream Cipher

Daniel J. Bernstein unveiled ChaCha20, a high-speed stream cipher, as an enhanced Salsa20 in 2008. Its design focuses on efficient, constant-time operations suitable for both software and hardware environments. The IETF standardized ChaCha20 together with the Poly1305 authenticator in RFC 7539, forming the basis of the *ChaCha20-Poly1305* authenticated encryption scheme used in TLS 1.3, SSH, and VPN protocols such as WireGuard.

A. Theoretical Overview

1) Cipher Structure: ChaCha20 initializes a 16-word (512-bit) state matrix using a 256-bit key, a 96-bit nonce, and a 32-bit block counter. Four constant words, eight key words, one counterword, and three nonce words make up the matrix. The cipher uses the following in 20 rounds (10 diagonal and

10 column rounds). *quarter-round* function on 32-bit words (a, b, c, d):

$$\begin{aligned} a &= a + b; \quad & d &= d \oplus a; \quad & d &= \text{ROTL}(d, 16) \\ c &= c + d; \quad & b &= b \oplus c; \quad & b &= \text{ROTL}(b, 12) \\ a &= a + b; \quad & d &= d \oplus a; \quad & d &= \text{ROTL}(d, 8) \\ c &= c + d; \quad & b &= b \oplus c; \quad & b &= \text{ROTL}(b, 7) \end{aligned}$$

Each block generates 64 bytes of keystream, which are XORed with plaintext to produce ciphertext. Because it uses only integer addition, XOR, and rotation, ChaCha20 avoids timing-dependent table lookups, making it secure against timing and cache-based side-channel attacks.

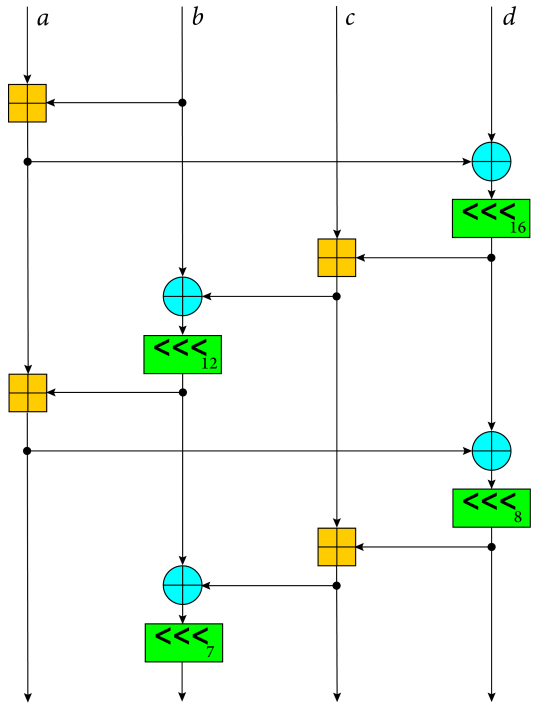

Fig. 3. ChaCha20 Quarter-Round Function. The four 32-bit words (a, b, c, d) are mixed using additions, XORs, and rotations.

2) Authenticated Encryption: ChaCha20-Poly1305: The ChaCha20 cipher is paired with the Poly1305 one-time authenticator to form an AEAD (Authenticated Encryption with Associated Data) construction. Poly1305 uses a one-time key derived from ChaCha20 to authenticate ciphertext and associated data, providing both confidentiality and integrity.

B. Performance Analysis

The performance of ChaCha20 depends heavily on the target platform. Unlike AES, it does not require hardware acceleration to achieve high speeds, making it especially efficient on devices lacking AES-NI or ARMv8 AES instructions.

- **Without AES Hardware Acceleration:** On mobile and general-purpose CPUs, ChaCha20 performs significantly better than AES-GCM (up to 3× faster on ARM, according to Cloudflare benchmarks).
- **With AES Hardware Acceleration:** Because of AES-NI and specialized hardware support, especially on x86-64 processors, AES-GCM is faster.
- **Energy Efficiency:** ChaCha20 reduces latency and power consumption on mobile and IoT devices by using less energy and running in constant time.

C. Security Considerations

ChaCha20 is resistant to all known practical attacks and offers 256-bit key security. Constant-time implementations are made easier by its simple layout, which reduces side-channel vulnerabilities. Because nonce reuse compromises the confidentiality of the stream cipher, the only significant security requirement is to ensure unique nonce usage per key.

VI. Methodology and Experimental Design

The implementation language, libraries, timing/measurement plan, and evaluation metrics that will be obtained in order to ensure predictable and comparable outcomes are described in this section.

A. Implementation Language and Libraries

For portability and quick development, the benchmark implementation will be written in **Python**.

B. Timing and Measurement Strategy

- **Granularity:** Calculate and report end-to-end totals as well as per-operation timings for symmetric encryption, symmetric decryption, key pair generation, and key exchange/wrapping.
- **Payload sizes:** Examine various payload sizes to see how symmetric costs predominate at scale.
- **Throughput:** Calculate MB/s for both symmetric encryption and end-to-end scenarios.

C. Metrics for Evaluation

The metrics listed below will be measured for every hybrid scheme and payload size:

- **Key generation time:** TIt's time to create asymmetric key pairs, if necessary.
- **Key exchange / wrapping time:** A symmetric key needs to be derived or wrapped.
- **Symmetric encryption time:** It's time to encrypt the payload with ChaCha20-Poly1305 or AES-GCM.
- **Symmetric decryption time:** The payload needs to be verified and decrypted.
- **Total end-to-end time:** Asymmetric setup and symmetric encryption/decryption are included in the total time.
- **Throughput:** For both end-to-end operations and the symmetric encryption stage, effective MBs were processed.

VII. Experimental Methodology

To empirically validate the theoretical performance differences discussed in previous sections, a comprehensive benchmarking suite was developed. The experiments were conducted to evaluate both asymmetric key generation costs and symmetric encryption throughput across various payload sizes.

A. Implementation Details

The benchmarking suite was implemented in **Python 3** utilizing the `cryptography` library. This library provides low-level cryptographic algorithms and binds to OpenSSL, ensuring that the performance reflects production-grade implementations rather than pure Python overhead.

1) Algorithms Tested:

- **Asymmetric Schemes:**
  - **RSA:** Key sizes of 2048, 3072, and 4096 bits.
  - **ECC (NIST Curves):** SECP256R1, SECP384R1, and SECP521R1.
- **Symmetric Schemes:**
  - **AES-256-GCM:** A widespread block cipher mode utilizing hardware acceleration where available.
  - **ChaCha20-Poly1305:** A software-friendly stream cipher with authenticated encryption.

B. Test Scenarios

The benchmark was divided into two distinct phases:

Phase 1: Asymmetric Key Generation. We measured the time required to generate private/public key pairs.

Phase 2: Symmetric Throughput. We simulated data transmission by encrypting random payloads of varying sizes, representing real-world use cases:

- **Tiny (64 Bytes):** Representative of handshake packets or authentication tokens.
- **Small (512 Bytes):** Representative of typical instant messaging payloads.
- **Medium (4 KB):** Representative of API responses or JSON objects.
- **Large (1 MB - 50 MB):** Representative of file transfers, images, or video streaming.

VIII. Experimental Results

The experimental data obtained from the Python benchmarking suite demonstrates distinct performance characteristics between the algorithms.

A. Asymmetric Key Generation

The performance gap between RSA and ECC was substantial. As shown in the generated data, increasing the key size for RSA resulted in a non-linear increase in generation time.

- **RSA Performance:** RSA-2048 required approximately 0.043 seconds, while RSA-4096 took significantly longer at 0.494 seconds. This confirms that RSA is computationally expensive at higher security levels.
- **ECC Performance:** ECC remained consistently fast across all curves. Even the strongest curve tested, SECP521, required only 0.0024 seconds. The standard SECP256 curve was orders of magnitude faster than RSA (0.000027 seconds).

B. Symmetric Encryption Throughput

For bulk data encryption, both algorithms performed exceptionally well, achieving speeds over 1 GB/s.

- **AES-256-GCM:** This algorithm achieved the highest throughput, peaking at approximately **1,650 MB/s** for 10 MB payloads. This performance indicates the effective utilization of hardware acceleration (AES-NI) present in the testing environment.
- **ChaCha20-Poly1305:** While slightly slower than the hardware-accelerated AES, the ChaCha20 stream cipher proved highly efficient, maintaining an average throughput of **0,943 MB/s** for large files (50 MB).

C. Latency on Small Payloads

For small payloads (64 bytes to 4 KB), representing handshake packets or chat messages, the latency difference was negligible. Both algorithms completed operations in approximately 3 to 5 microseconds (10^{-6} seconds), indicating that neither algorithm creates a bottleneck for real-time communications.

IX. Conclusion

This comparative overview highlights the evolution of cryptographic standards from RSA/AES to modern alternatives like ECC/ChaCha20.

Based on the theoretical analysis and experimental results, we conclude:

1) **Public Key Cryptography: ECC** is the superior choice for modern applications. It offers equivalent security to RSA with significantly smaller keys and faster generation times, making it ideal for mobile devices, IoT, and high-frequency handshakes (e.g., TLS).
2) **Symmetric Cryptography: AES-256-GCM** remains the gold standard for server environments where hardware acceleration is available. However, **ChaCha20-Poly1305** is a valid alternative, offering comparable security and excellent performance in software, particularly for devices lacking specialized AES hardware.

Ultimately, the choice of algorithm depends on the specific hardware environment and the type of data being protected.

References

[1] *Advanced Encryption Standard*. URL: https : / / en . wikipedia.org/wiki/Advanced_Encryption_Standard.

[2] Daniel J. Bernstein. *ChaCha, a variant of Salsa20*. https://cr.yp.to/chacha.html. 2008.

[3] Joppe W Bos et al. "Elliptic Curve Cryptography in Practice". In: *Financial Cryptography and Data Security* (2013).

[4] *ChaCha20-Poly1305*. URL: https://en.wikipedia.org/wiki/ChaCha20-Poly1305.

[5] *Elliptic-curve cryptography*. URL: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography#cite_note-2.

[6] Rahoul Ganesh et al. "A panoramic survey of the advanced encryption standard: from architecture to security analysis, key management, real-world applications, and post-quantum challenges". In: *International Journal of Information Security* (2025). DOI: 10.1007/s10207-025-01116-x.

[7] Ahmed Othman Khalaf et al. "Comparison between RSA, ECC & NTRU Algorithms". In: *International Journal of Engineering Research and Advanced Technology (IJERAT)* (2019).

[8] Ahmed Othman Khalaf et al. "Subject Review: Comparison between RSA, ECC NTRU Algorithms". In: *International Journal of Engineering Research and Advanced Technology (ijerat)* (2019). DOI: 10.31695/IJERAT.2019.3582.

[9] Adam Langley, Mark Hamburg, and Stephen Turner. *ChaCha20 and Poly1305 for IETF Protocols*. https://www.rfc-editor.org/info/rfc7539. RFC 7539. IETF, 2015.

[10] Bhanu Prakash et al. "A Numerical and Security Analysis of RSA: From Classical Encryption to Post-Quantum Strategies". In: *AI and Intelligent Systems: Engineering, Medicine Society (AIISEMS)* (2024).

[11] *RSA cryptosystem*. URL: https://en.wikipedia.org/wiki/RSA_cryptosystem.