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

***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 a fundamental discipline within information security that enables the protection of data against unauthorized access, tampering, and forgery. It applies mathematical algorithms to transform readable data (plaintext) into an unreadable form (ciphertext), ensuring that only authorized entities can interpret the original information. Cryptography underpins secure communication, authentication, and data integrity across modern network systems. The main objectives of cryptography are: confidentiality, integrity, authentication, and non-repudiation. The implementaions will be uploaded on github at this link: https://github.com/Luke-MT/Network-and-Information-Systems-Security-Project

A. Main Goals of Cryptography

- **Confidentiality:** Ensures that only authorized parties can access transmitted or stored information. Data is encrypted using secret keys so that even if it is intercepted, it remains unreadable without the correct key.
- **Integrity:** Protects data from unauthorized modification. Hash functions and message authentication codes (MACs) are used to verify that information has not 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 a sender cannot deny sending a message or performing an action. This is achieved using digital signatures that provide undeniable proof of origin.

B. Symmetric vs. Asymmetric Encryption

The following table contrasts the two primary approaches to encryption.

TABLE I
Comparison of Symmetric and Asymmetric Encryption

Feature	Symmetric Encryption	Asymmetric Encryption
Key Structure	Uses a single shared secret key for both encryption and decryption.	Uses a pair of mathematically related keys – a public key and a private key.
Examples	AES, ChaCha20	RSA, ECC
Speed	Faster and suitable for bulk data encryption.	Slower due to 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 project, AES represents the symmetric encryption approach, while RSA and ECC demonstrate asymmetric encryption.

II. RSA: Rivest–Shamir–Adleman

RSA, introduced by Rivest, Shamir, and Adleman in 1977, remains a cornerstone of asymmetric cryptography. Security rests on the presumed hardness of factoring the product of two large primes $n = pq$. Despite the emergence of more efficient public-key primitives, RSA persists in practice due to its simplicity, maturity, and extensive ecosystem support. This paper frames RSA from a theoretical performance perspective to complement a broader group comparison with AES, ChaCha20, and ECC.

A. Mathematical Foundations

Let p and q be 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 an advanced form of public-key cryptography (PKC) that relies on the algebraic structure of elliptic curves over finite fields. Proposed independently by Neal Koblitz and Victor Miller in the mid-1980s, ECC provides a high level of security with significantly **smaller key sizes** compared to older public-key systems like RSA. This efficiency makes ECC ideal for resource-constrained environments such as mobile devices and the Internet of Things (IoT). The security of ECC is founded on the presumed intractability of 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 add two distinct points P and Q, one draws a straight line through them. This line will intersect the curve at a third point, 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.

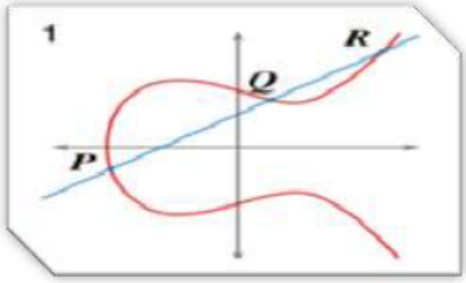

Fig. 1. Geometric representation of point addition ($P+Q = R$) on an elliptic curve.

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)**, used to establish a shared secret key between two parties. This shared secret is then used with a symmetric cipher for data encryption.

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 derivation function (KDF) to produce a key suitable for a symmetric encryption algorithm.

D. Advantages and Practical Applications

The primary advantage of ECC over its predecessors is its ability to provide equivalent security with much smaller key sizes, leading to significant performance benefits.

1) Efficiency and Key Size:

- **Faster Computations:** Key generation and signing operations are significantly quicker.
- **Lower Power Consumption:** Ideal for battery-powered devices like smartphones and IoT sensors.
- **Reduced Bandwidth and Storage:** Smaller keys and signatures require less space and data transfer.

2) Real-World Use Cases: ECC is the foundation of modern digital security and is used in a wide range of applications:

- **Web Security:** The Transport Layer Security (TLS) protocol, which secures HTTPS connections, widely uses ECDH for its key exchange mechanism (specifically, ECDHE).
- **Cryptocurrencies:** Bitcoin, Ethereum, and many other cryptocurrencies use the Elliptic Curve Digital Signature Algorithm (ECDSA) to secure wallets and authorize 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 symmetric-key block cipher standardized by the U.S. National Institute of Standards and Technology (NIST) in 2001. Developed by Belgian cryptographers Joan Daemen and Vincent Rijmen, the algorithm, originally named **Rijndael**, was selected through an open, multi-year competition to replace the aging Data Encryption Standard (DES). Unlike asymmetric cryptosystems (like ECC or RSA) that use different keys for encryption and decryption, AES is a **symmetric** algorithm, meaning the same secret key is used for both processes. It operates on fixed-size blocks of data specifically, 128-bit blocks and is available in three key 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 processes a 128-bit block of plaintext by arranging it into a 4x4 matrix of bytes called the **State**. The algorithm then iteratively applies a series of mathematical transformations to this State over multiple rounds. Each round is designed to introduce confusion and diffusion, two key principles of modern cryptography that obscure the relationship between the plaintext, the ciphertext, and the key.

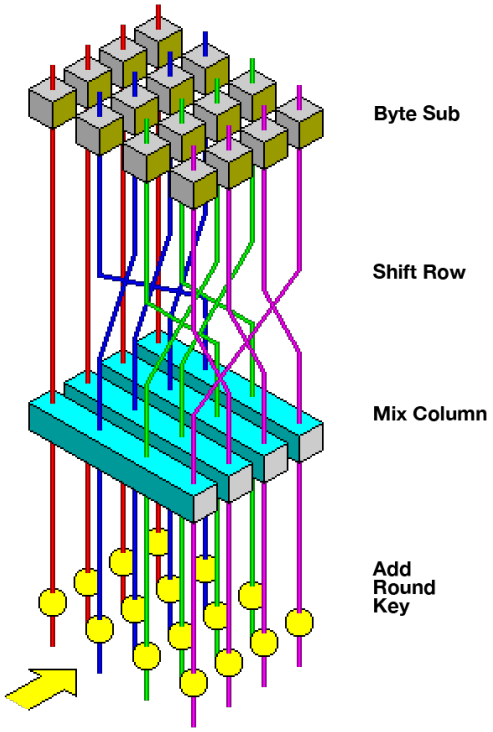

Fig. 2. Overview of the AES Encryption Process

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

1) The Four Transformation Layers:

1) **`SubBytes` (Substitution):** This is a non-linear byte substitution step where each byte of the State is replaced with another according to a predefined lookup table known as 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):** In this step, the bytes in each row of the State are cyclically shifted to the left. The first row is not shifted, the second is shifted by one byte, 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 operates on each column of the State individually, combining the four bytes in each column using a specific mathematical operation over a finite field (Galois Field $GF(2^8)$). The `MixColumns` step, along 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):** In this final step of the round, the State is combined with a portion of the expanded key, 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 plaintext State is XORed with the first round key.
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

ChaCha20 is a high-speed stream cipher introduced by Daniel J. Bernstein in 2008 as an improved version of Salsa20. 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 uses a 256-bit key, a 96-bit nonce, and a 32-bit block counter to initialize a 16-word (512-bit) state matrix. The matrix includes four constant words, eight key words, one counter word, and three nonce words. The cipher operates through 20 rounds (10 column and 10 diagonal rounds), applying the following *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.

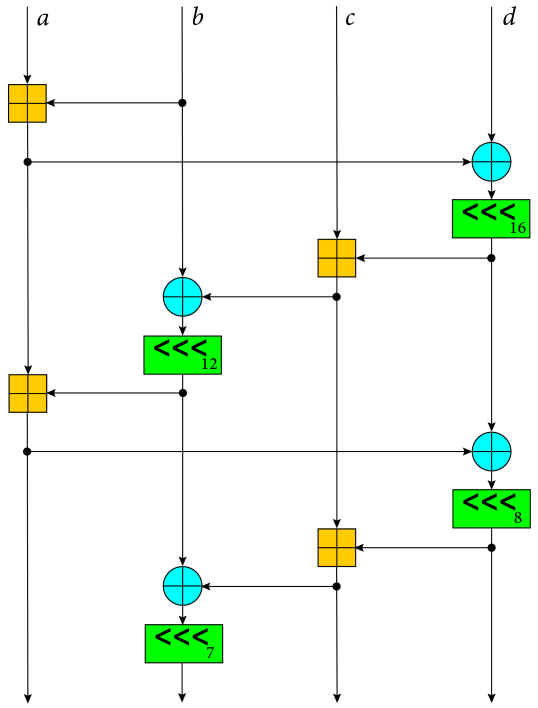

Fig. 3. ChaCha20 Quarter-Round Function. The four 32-bit words (a, b, c, d) are mixed using additions, XORs, and rotations.

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:** ChaCha20 significantly outperforms AES-GCM on general-purpose CPUs and mobile processors (up to 3× faster on ARM, according to Cloudflare benchmarks).
- **With AES Hardware Acceleration:** AES-GCM is faster due to AES-NI and dedicated hardware support, particularly on x86-64 processors.
- **Energy Efficiency:** On mobile and IoT devices, ChaCha20 consumes less energy and executes in constant time, reducing both latency and power consumption.

C. Security Considerations

ChaCha20 provides 256-bit key security and resists all known practical attacks. Its design simplicity facilitates constant-time implementations, minimizing side-channel vulnerabilities. The only major security requirement is ensuring unique nonce usage per key, as nonce reuse compromises the stream cipher's confidentiality.

VI. Methodology and Experimental Design

This section details the implementation language, libraries, the timing/measurement strategy, and the evaluation metrics that will be collected to ensure results are reproducible and comparable.

A. Implementation Language and Libraries

The benchmark implementation will be written in **Python** for portability and rapid development.

Additional tooling and practices:

- Use `os.urandom()` for cryptographic randomness.
- Use `time.perf_counter()` for high-resolution timing.
- Run each measured operation multiple times (e.g., 50–200 trials) and compute mean and standard deviation.
- Include a warm-up phase to mitigate cold-start effects and library initialization costs.

B. Timing and Measurement Strategy

- **Granularity:** Measure and report per-operation timings (keypair generation, key exchange/wrapping, symmetric encryption, symmetric decryption) and end-to-end totals.
- **Payload sizes:** Test multiple payload sizes (e.g., 1 KB, 64 KB, 1 MB) to observe how symmetric costs dominate at scale.
- **Repetitions:** Repeat each measurement for many trials to capture variance; report mean, median, and standard deviation.
- **Throughput:** Derive MB/s for symmetric encryption and for combined end-to-end scenarios.

C. Metrics for Evaluation

The following metrics will be collected for each hybrid scheme and payload size:

- **Key generation time:** Time to generate asymmetric key pairs (where applicable).
- **Key exchange / wrapping time:** Time to derive or wrap a symmetric key.
- **Symmetric encryption time:** Time to encrypt the payload using AES-GCM or ChaCha20-Poly1305.
- **Symmetric decryption time:** Time to decrypt and verify the payload.
- **Total end-to-end time:** Aggregate time including asymmetric setup and symmetric encryption/decryption.
- **Throughput:** Effective MB/s processed for the symmetric encryption stage and for end-to-end operations.
- **Statistical measures:** Mean, median, and standard deviation across repeated trials.

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.