# Performance Analysis of Encryption and Decryption Techniques

*Exploring Speed, Efficiency, and Use Cases Across Key Sizes, File Sizes, and Batch Processing*

Report By: Beck Schemenauer

## 1. Background

Encryption is the most important factor in securing sensitive data in modern applications. As data transmission grows in scale and complexity, encryption methods are created or adapted to be more secure and handle more information. Evaluating their behavior under varying conditions allows for understanding the best applications of each technique.

This study aims to evaluate the performance of a few encryption and decryption techniques, focusing on speed, sustained performance, and optimal use cases.. The methods analyzed include Advanced Encryption Standard (AES) in both Electronic Codebook (ECB) and Cipher Block Chaining (CBC) modes, two widely-used stream ciphers (RC4 and ChaCha20), and Elliptic Curve Cryptography (ECC) for secure key generation, paired with CBC encryption.

These techniques were tested across a range of file sizes (1 MB to 1 GB), as well as different AES key lengths (128-bit, 192-bit, and 256-bit). Additionally each algorithm was tested under a simulated batch processing scenario. The findings will help determine the optimal application contexts for these encryption techniques, considering efficiency, throughput, and scalability.

## 2. Methodology

### 2.1 Experimental Setup

The testing environment was configured to evaluate encryption and decryption throughput for various file sizes and key lengths. The experiments were run on a Windows 11 x64 system with an AMD Ryzen 7 4800H processor (8 cores, 16 threads) and 16 GB of RAM. PyCryptodome and Python were used to implement the encryption This setup remained consistent during testing.

Key configurations included:

- **File Sizes**: Files of 1 MB, 10 MB, 100 MB, and 1,000 MB were tested.
- **Key Lengths**: For ECB, CBC, and RC4 modes, key sizes of 128 bits (16 bytes), 192 bits (24 bytes), and 256 bits (32 bytes) were used.
- **Iterations**: Each test was repeated 100 times per file size and key length to compute average throughput and account for variability.

## 2.2 Encryption Techniques Used

**AES (Advanced Encryption Standard) in ECB Mode:** A block cipher that operates on fixed-size blocks of data (16-byte block size). The ECB mode encrypts each block of plaintext independently using the same key, producing a corresponding block of ciphertext without any chaining between blocks.

**AES (Advanced Encryption Standard) in CBC Mode:** A block cipher that operates on fixed-size blocks of data (16-byte block size). In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This process requires an initialization vector (IV) for the first block and ensures that each block's encryption depends on the preceding one.

**Stream Cipher:** A cryptographic algorithm that encrypts plaintext one bit or byte at a time, using a pseudorandom keystream that is XORed with the plaintext to produce ciphertext.

**RC4:** A stream cipher that encrypts data one byte at a time. It generates a pseudorandom keystream that is XORed with the plaintext to produce ciphertext, using a key-scheduling algorithm and a pseudorandom number generator.

**ChaCha20:** A stream cipher that operates by generating a pseudorandom key stream from a 256-bit key and a 96-bit nonce. The keystream is XORed with the plaintext to produce ciphertext. It uses a series of mathematical operations, including addition, rotation, and XOR, to ensure randomness.

**ECC with CBC:** Combines Elliptic Curve Cryptography (ECC) for generating and securely exchanging cryptographic keys with AES in CBC mode for encrypting data. ECC uses the mathematics of elliptic curves to produce compact, efficient keys for encryption and decryption.

## 2.3 Test Scenarios

### File Size Variations

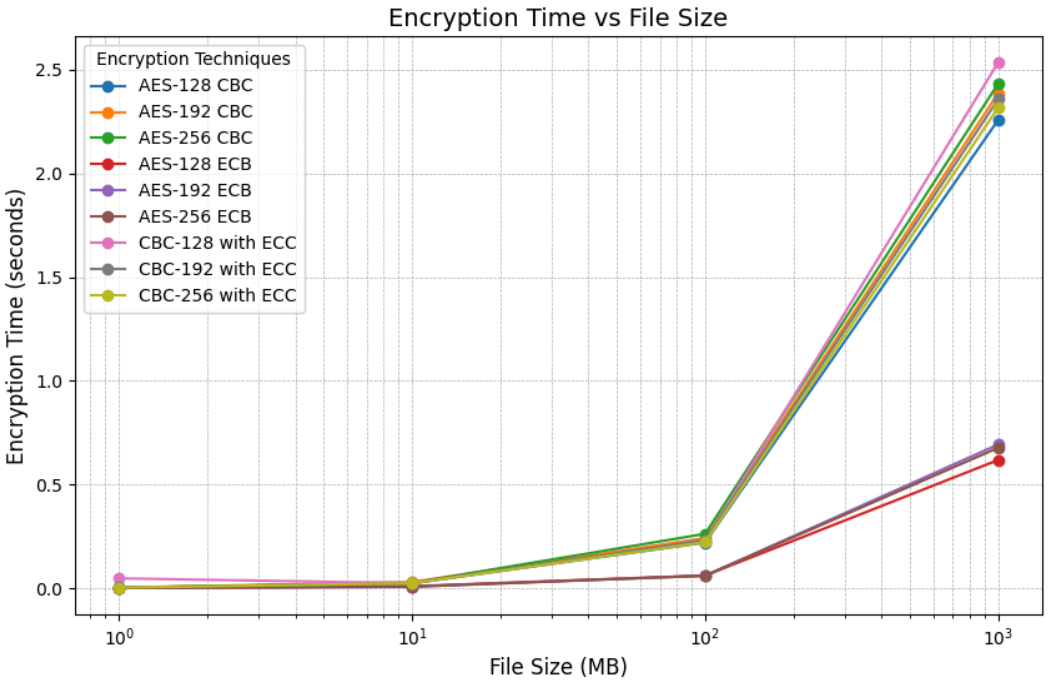
The experiments analyzed the effect of file size on encryption and decryption throughput. Files of varying sizes were encrypted and decrypted to assess how the algorithms scaled with larger data. For instance, while throughput generally increases with larger files, computational overhead such as padding and initialization can impact the results for smaller files.

**Key Size Variations**

For AES in CBC mode, different key lengths (128-bit, 192-bit, 256-bit) were tested to evaluate the trade-offs between security and performance. Larger keys provide stronger security due to increased resistance to brute-force attacks but may have a slight impact on speed. Results showed that encryption and decryption throughput was similar across key sizes, with longer keys incurring negligible time penalties.

## 3. Results and Analysis

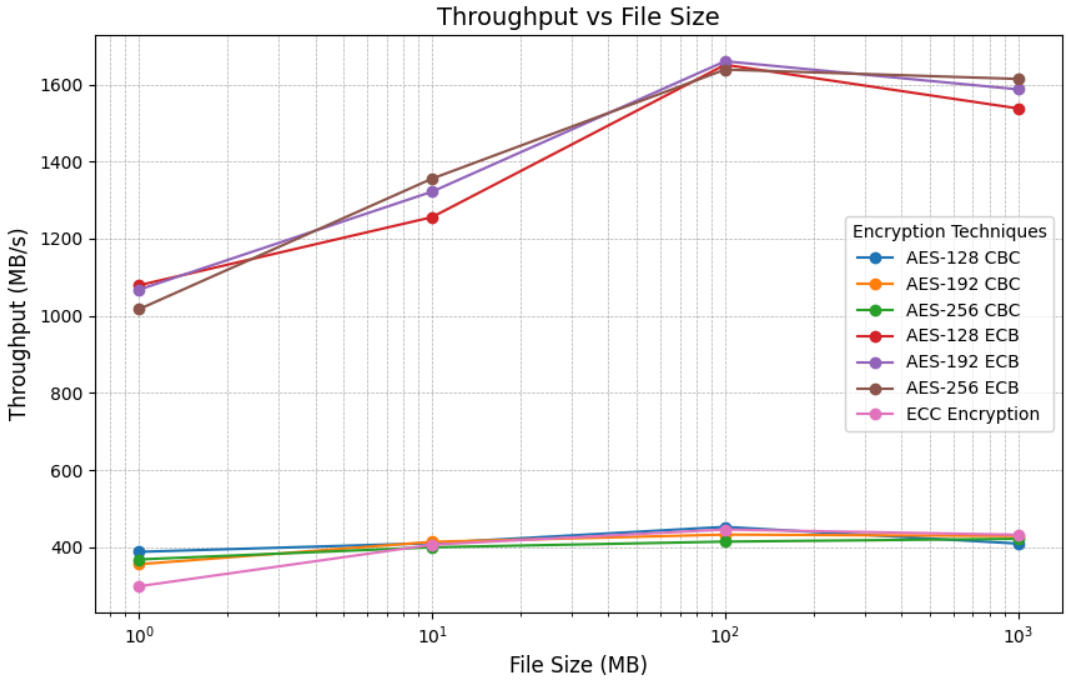
### 3.1 AES: ECB vs. CBC (Including ECC)




(Figure 1: Encryption Time vs File Size for AES modes and ECC)

The above graph shows encryption time versus file size for different AES encryption methods, along with ECC key generation combined with CBC. CBC, with or without ECC key generation, took significantly longer than ECB modes, highlighting the trade-off between performance and security. While ECB is faster, taking around 0.6 seconds, it is less secure because it preserves patterns in the plaintext, making it unsuitable for sensitive or repetitive data, such as photos. In contrast, CBC, which took approximately 2.4 seconds, provides better security by chaining encryption blocks, ensuring unique ciphertexts even for identical plaintext blocks. Adding ECC key generation to CBC slightly increases computational overhead but enhances security by incorporating asymmetric encryption for key exchange, while CBC itself remains a symmetric encryption mode.

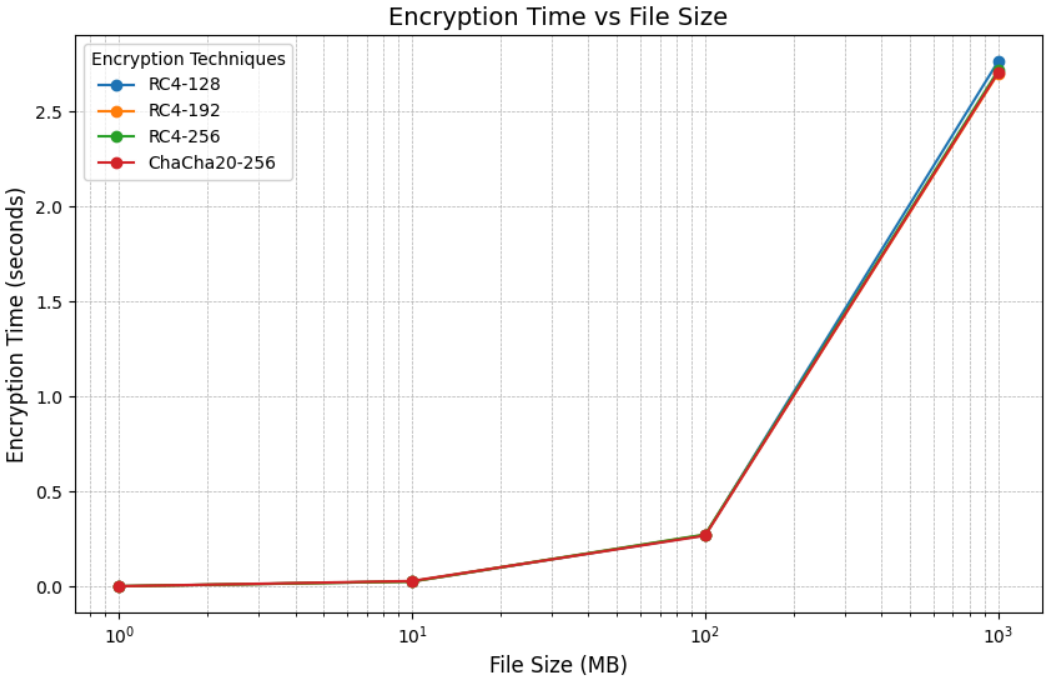
When choosing encryption methods, it is generally best to use longer keys, as they significantly increase resistance to brute-force attacks with minimal impact on performance. For encrypting large volumes of non-sensitive, non-repetitive data, ECB with a 256-bit key can be a practical choice due to its speed. However, for sensitive data or when security is a priority, CBC-256 combined with ECC key generation is the better option, as it offers robust protection against potential attacks.




(Figure 2: Throughput vs File Size for AES modes and ECC)

The additional graph shows throughput measurements in MB/s for different file sizes, collected by running each file 100 times. Consistent with the earlier findings, CBC modes exhibit significantly lower throughput compared to ECB modes, which are nearly four times faster. While this graph does not introduce new insights regarding use cases, it complements the previous results by providing a clearer visualization of performance differences. Throughput, measured as data processed per second, minute, or hour, offers a more practical perspective for evaluating encryption methods in large-scale applications.

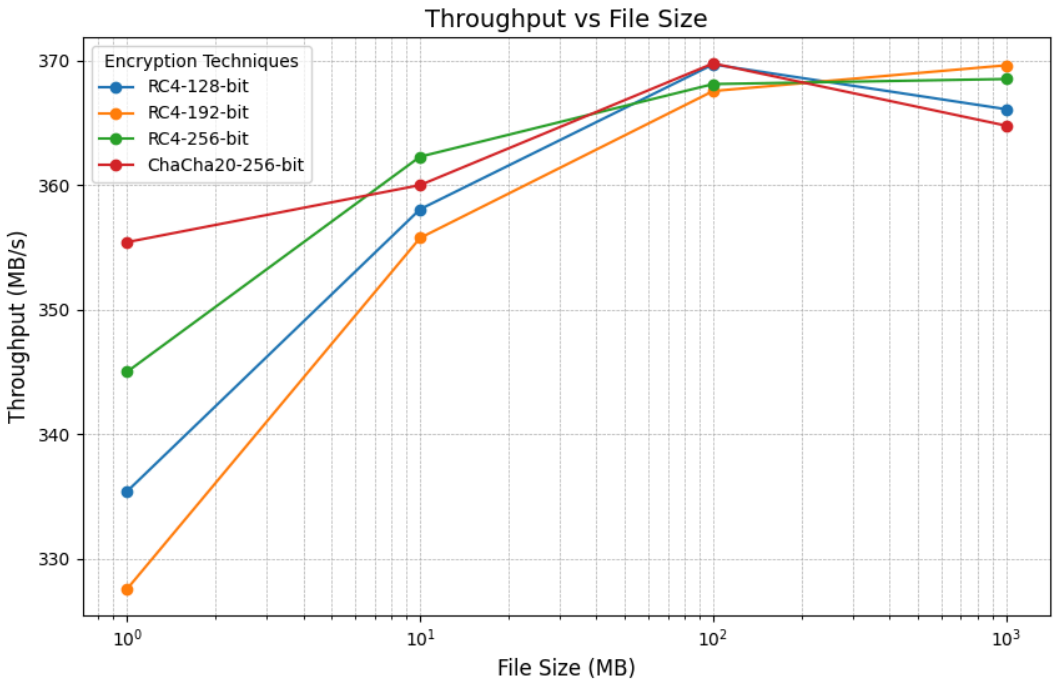
### 3.2 Stream Ciphers: RC4 vs. ChaCha20




(Figure 3: Encryption Time vs File Size for Stream Cipher modes)

The above graph displays encryption time versus file size for different stream cipher modes, including RC4 with 128-bit, 192-bit, and 256-bit keys, as well as ChaCha20. The results show that all modes performed similarly, with encryption times around 2.7 seconds.

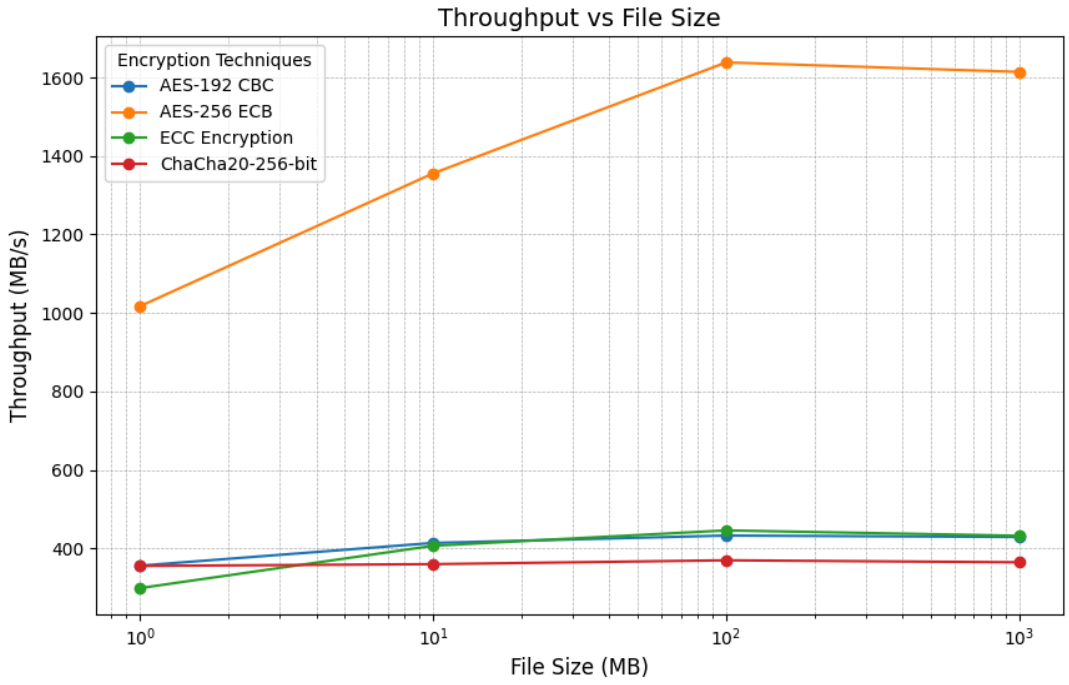
Despite the equivalence in speeds though, RC4 is deprecated due to known vulnerabilities in its key scheduling and output streams, making it susceptible to various attacks. On the other hand, ChaCha20 is more secure, offering robust resistance to attacks through more modern design and optimizations. This makes ChaCha20 a better option for applications where both speed and strong security are necessary. However RC4 may still be useful for non-critical systems or older systems where security risks are minimal.




(Figure 4: Throughput vs File Size for Stream Cipher modes)

Similar to the discussion on AES encryption modes, the throughput does not significantly change the perspective of use cases. However, it is noteworthy that ChaCha20 maintained a consistently high throughput across different file sizes, likely due to its newer and more efficient design. ChaCha20 is optimized for modern processors, using constant-time operations and avoiding the inefficiencies in RC4's key scheduling and output generation, which contribute to RC4's more logarithmic performance curve. Regardless of the deprecation and security risks of RC4, the more efficient throughput and stability makes ChaCha20 a better choice in terms of stream ciphers.

## 4. Overall Use Cases




(Figure 5: Top Throughputs Across Various Modes)

The graph compares the top throughputs across CBC, ECC with CBC, ChaCha20, and ECB. The first four mentioned all exhibit similar throughputs, showing their comparable performance. In contrast, ECB achieves a throughput roughly four times as high as the other methods, making it the fastest among them. However, this performance advantage comes at the cost of significantly reduced security, as ECB is vulnerable to pattern preservation in ciphertexts. When this is not a concern, and when dealing with large amounts of data, ECB is by far the best technique. However, when security is a concern, due to similar performances, CBC with ECC is the best option as it combines the strong data encryption of CBC with the secure key exchange of ECC, providing the highest level of protection.

---

## 5. Conclusion

- Recap of findings and their significance.
- Final recommendations for encryption method selection based on specific operational needs.