HSM-based Performance Analysis of Cryptographic Algorithms

Ruthvik V, S Sricharan, Samarth Jayanth
Indu Radhakrishnan
Department of Electronics and Communication Engineering, PES University, Bengaluru 560085, India
Department of Computer Science and Engineering, PES University, Bengaluru 560085, India
Emails: v.ruthvik37@gmail.com, ssricharan2003@gmail.com, samarth_26@yahoo.com, induradhakrishnan@pes.edu

I. Introduction

In the current world where cyber threats and data breaches are on the rise, the need for robust security solutions has become prominent. Hence, the adoption of HSMs has increased a lot because of their strong security features. A hardware security module is a tamper- and intrusion-resistant, highly trusted physical computing device that performs all major cryptographic operations. The sole purpose of an HSM is to conceal and protect cryptographic data [1]. The core function of HSMs is key generation, which involves securely creating cryptographic keys that are used for encryption and decryption purposes.

HSM's provide a high degree of security due to the isolation of cryptographic operations such as key generation and management functions from the other parts of the system, making it difficult for malicious operators to access unauthorized data and tamper with sensitive information. For these reasons and more, HSMs are considered the Root of Trust in many organizations. The Root of Trust is a source in a cryptographic system that can be relied upon at all times. The strict security measures used within an HSM allow it to be the perfect Root of Trust in any organization's security infrastructure [2].

As a result, businesses across various sectors rely on HSMs for important security operations.
For instance, financial services rely on them to provide encryption of transaction data, manage payment card information and secure online banking systems. In healthcare, HSMs help in safeguarding patient records and ensuring HIPAA compliance. HSMs encrypt classified information and secure communication channels used by the government and defense industries. HSMs also serve as guarantors of secure online transactions among e-commerce platforms while at the same time protecting valuable customer data [3]. Hence, HSMs integrated into various security architectures have become a solution for providing a trusted environment capable of executing cryptographic operations and managing cryptographic keys.

Since safety and security have emerged as a critical feature in nearly all areas of communication, while sending a message to someone over an insecure channel along with internet we have to provide confidentiality, integrity, authenticity and non-repudiation [4] especially in applications that involve personal information or any transactions. This process involves several critical components, with key generation and encryption being vital.

The process of converting plain-text into cipher-text is known as encryption, while the reverse process where cipher-text is converted back to plain-text is known as decryption. These processes can be achieved through two methods: symmetric key and asymmetric key cryptography. Symmetric key cryptography technique uses the same key for encryption and decryption of plain-text and cipher-text. In asymmetric key encryption cryptography there is use of a pair of keys called public and private keys to encrypt the plain-text and decrypt the cipher-text respectively; it's also known as public key cryptography. In uneven key encryption public keys are shared publicly with everybody, however private keys are best recognized by the owner [5].

In this paper, we theoretically analyse the cryptographic algorithms [RSA, 3DES and AES] and also implement C++ programs for comparing them practically based on performance metrics such as encryption Throughput, decryption Throughput, key generation time, memory usage and CPU usage.

II. Background and Related Work

SoftHSM2 is an open source software implementation of a cryptographic store. Using PKCS#11 APIs, it emulates an HSM environment which is usable for development and testing. With SoftHSM2 being an emulation of the actual HSM, it is possible for developers and organizations to consider the advantages of using HSMs without the associated costs and logistical challenges.

PKCS#11 (Public-Key Cryptography Standards #11) is a well known standard for a platform independent interface for cryptographic tokens like HSM and smart cards. This API is used to enable applications that require cryptographic services to run without having to know the details of the underlying hardware. Thus, it serves as an interface for managing keys as well as for encryption and decryption processes, which makes it a foundation for building secure applications.

In the PKCS#11 framework there are two basic concepts called slots and tokens. A slot can be a physical or logical reader that can hold a token. A token, on the other hand, is the physical entity or gadget that can hold the cryptographic keys and can also process the cryptographic operations. In the case

of SoftHSM2, the slots are virtual within the software and the tokens are also virtual which mimic the functionality of physical cryptographic tokens. This setup enables the developers to simulate the functionality of hardware HSMs which helps them develop and test the applications in a more effective way.

In this context, virtual HSMs like SoftHSM2 play a critical role in providing a flexible and cost-effective solution for generating, storing, and managing keys securely while facilitating encryption and decryption processes within a software based environment.

Securing the keys in a cryptographic system is critical to maintaining a secure system and hence managing them is a big challenge. That's where HSMs come in. They manage all aspects of a cryptography key's life-cycle, including the following six steps:
Provisioning: Generation of keys by a true random number generator or by cyptographic algorithms such as RSA, 3DES and AES.
Backup and storage: A copy of a key should be made and securely stored in case the key is compromised or lost.
Deployment: This involves installing the key in a cryptographic device such as an HSM.
Management: Keys are controlled and monitored based on industry standards and an organization's own internal policies.
Archiving: Decommissioned keys are put in offline, long-term storage for when they may be needed to access existing data that was encrypted with that key.
Disposal: Keys should be securely and permanently destroyed only after it is determined that they are no longer needed [6]. All the steps of cryptographic key life-cycle are shown in Figure 1.

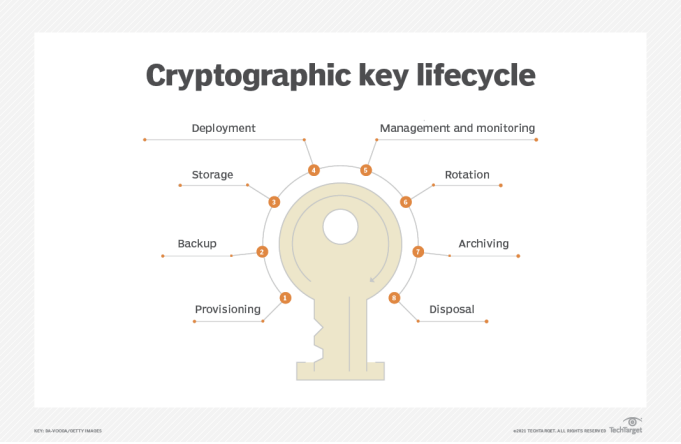

Figure 1: Cryptographic Key Life-cycle.

Selecting the right cryptographic algorithms for a specific application is crucial. Here are the main criteria we consider when choosing an algorithm:
Security Strength: The algorithm must provide a sufficient level of security to protect any sensitive data from being compromised.
Performance: To meet the demands of various applications, the algorithm should have efficient performance, especially in terms of speed and resource utilization.
Compliance: The algorithm must comply with relevant industry standards and regulations, such as FIPS 140-2/3 and PCI-DSS.
Use Case: We have to choose the correct algorithm that fits the specific use case because different algorithms are suited to different tasks, such as data encryption, digital signatures, or key exchange.

Given these criteria, we have decided to focus on three widely used cryptographic algorithms: 3DES, AES and RSA. These algorithms are chosen due to their extensive use in various applications and their proven security and performance characteristics.

A. *Rivest-Shamir-Adleman (RSA) Algorithm*

RSA stands for Rivest-Shamir-Adleman, an asymmetric-key encryption algorithm, first proposed in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman. The concept of RSA is based on the fact that large integers are difficult to factorize. It often finds huge applications in secure data transmission, generating digital signatures, and key exchange. RSA generates a public key for encryption and a private key for its decryption.

Given the public key $(n, e) = k_{pub}$ and the plain-text x, the Encryption function is:

$$y = e_{k_{pub}}(x) \equiv X^e \mod n$$

and where $x, y \in \mathbb{Z}_n$. [8]

RSA Decryption: Given the private key $d = k_{pr}$ [8] and the cipher-text y, the decryption function is:

$$x = d_{k_{pr}}(y) \equiv Y^d \mod n$$

and where $x, y \in \mathbb{Z}_n$. [8] Usually, these numbers x, y, n and d are very large, about 1024 bits in size [8]. In this process the private key number d is referred to as the decryption number and the public key number e is referred to as the encryption number [4], [8]. $\phi(n)$ is the Euler's totient function. The RSA key generation involves five steps illustrated below:

RSA Key Generation Output: public key: $k_{pub} = (n, e)$ and private key: $k_{pr} = (d)$

1) Choose two large primes p and q.
2) Compute $n = p \times q$.
3) Compute $\Phi(n) = (p-1)(q-1)$.
4) Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that $\gcd(e, \Phi(n)) = 1$.
5) Compute the private key d such that $d \cdot e \equiv 1 \mod \Phi(n)$ [8].

A message x is encrypted using the public key e of the recipient, and the result can be decrypted using the private key d, which will give back the original message x. This process is illustrated by the following equation:

$$d_{k_{pr}}(y) = d_{k_{pr}}(e_{k_{pub}}(x)) = x^{de} = x \mod n$$

The block diagram for RSA algorithm [11] is shown in Figure 2.

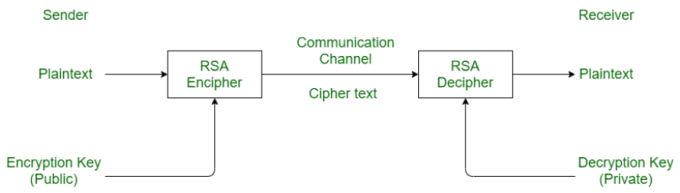

Figure 2: RSA Block Diagram.

B. Triple Data Encryption Standard (3DES) Algorithm

3DES is a symmetric key block cipher. The security of the original DES algorithm is improved by applying it three times to each block of data. Due to the additional safety provided by this method, it has been used in many applications where more security is needed than DES provides. 3DES triple-encrypts every block of plain-text using the DES cipher algorithm. Triple DES works with a packet of three 56-bit keys that is, a total of 168 bits. The triple DES process undergoes three stages: an encryption step with the first key, followed by a decryption step with the second key, and finally another encryption step with the third key. This triple application of DES makes 3DES much more secure compared to its predecessor and minimizes several vulnerabilities associated with the smaller key size of the single DES standard. The block diagram for 3DES algorithm [12] is shown in Figure 3.

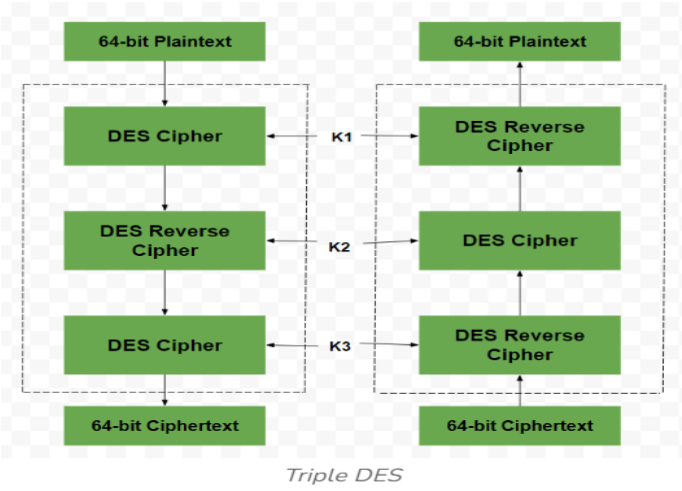

Figure 3: 3DES Block Diagram.

C. Advanced Encryption Standard (AES) Algorithm

AES is the Advanced Encryption Standard, established in 2001. It is a symmetric block cipher known for its speed and security with 128-bit block sizes and three key sizes of 128, 192, and 256 bits. It has wide applications in secure data encryption. AES represents a substitution of plain-text into the cipher-text under multiple substitution and permutation operations organized for several rounds according to the following conditions: 10 rounds for keys of 128 bits, 12 rounds for 192-bit keys, and finally 14 rounds for keys of 256 bits. Every round involves several steps: (i) SubBytes that is byte substitution based on an S-box; (ii) ShiftRows for permutation by rows; (iii) MixColumns for column mixing; (iv) AddRoundKey for bit-wise operation of the round key, directly from the encryption key. Decryption is the inverse process [13]. The block diagram for AES algorithm [13] is shown in Figures 4 and 5.

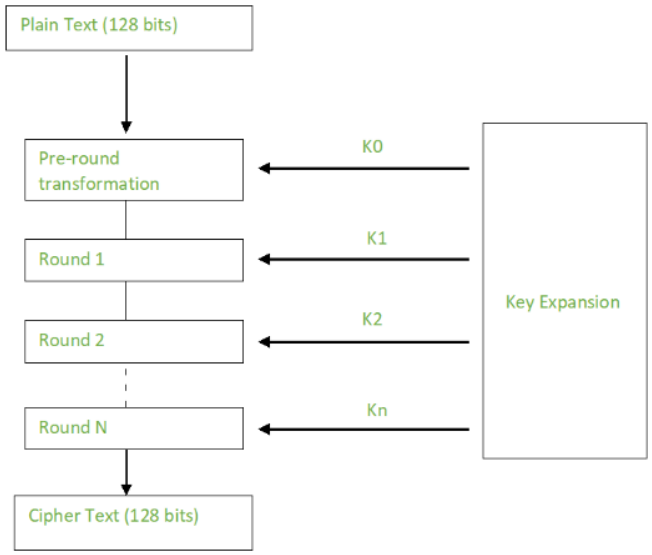

Figure 4: AES Block Diagram.

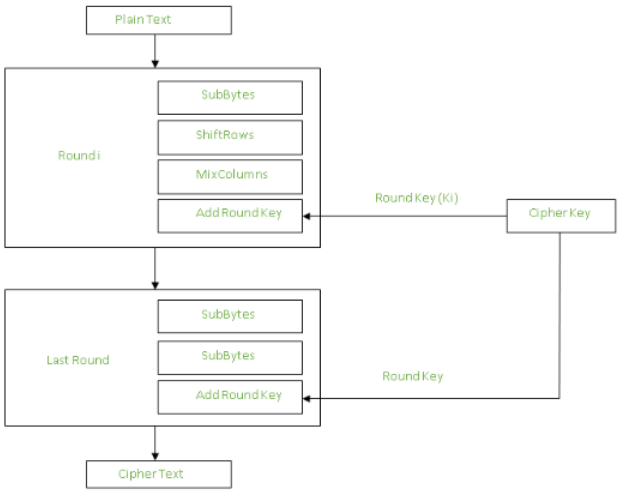

Figure 5: AES Block Diagram.

When comparing these algorithms by examining their block diagrams and methodologies, certain observations can be made. AES is recognized for its impressive speed and robust security. It is more complex than some older algorithms, but its security, efficiency and flexibility with varying key lengths makes it widely applicable. 3DES, on the other hand, enhances security through triple encryption while retaining compatibility with DES, with a relatively small effective key length. RSA stands out for its high security, yet it is slower than symmetric-key algorithms due to its extensive mathematical computations and its larger key sizes.

III. Methodology

In this section, we discuss the process followed to compare the cryptographic algorithms [RSA, 3DES and AES] based on performance metrics such as Encryption Throughput, Decryption Throughput, Key Generation Time, Memory Usage, CPU Usage using SoftHSM2. The steps involved are critical to ensuring the secure generation, management, and utilization of cryptographic keys and the encryption-decryption processes.

SoftHSM2 was employed to initialize slots and tokens. These slots and tokens are essential components in interacting with the virtual Hardware Security Module (HSM). Here, the slots represent logical entities that hold the tokens, which are the actual cryptographic modules responsible for storing and managing keys.

The process began with setting up the slots and tokens using SoftHSM2. Each slot was initialized to hold a token, which then simulated the functionality of a physical HSM. This setup allowed for a realistic emulation of hardware-based security within a software environment, making it possible to develop and test the cryptographic operations required for secure communication.

The C++ program was designed to perform several key tasks, leveraging the PKCS#11 API provided by SoftHSM2. The following steps outline the methodology:

Global Variables and Function Declarations :- At the top of the example, it declares several global variables: handles for the PKCS#11 library and session, slot ID, PIN for a slot, and buffers to hold encrypted and decrypted data. It also includes function declarations on loading the HSM library, freeing of resources, printing data in hexadecimal and plain text formats, connecting to and disconnecting from a slot, generating cryptographic keys, encrypting and decrypting data, measuring key generation, encryption, and decryption times, obtaining current memory and CPU usage, and calculating throughput for a given operation.

Loading the HSM Library :- The loadHSMLibrary function loads the PKCS#11 library identified by the P11_LIB environment variable, gets a function list from the library, and saves it in the global p11Func variable. If the loading of the library or the function list fails, the function prints an error message and exits the program.

Connection and disconnection of slots :- The connectToSlot function initializes the PKCS#11 library, opens a session with the given slot ID, and logs in with the given PIN. The disconnectFromSlot function logs out of the session, closes the session, and finalizes the PKCS#11 library.

Generation of Cryptographic Keys :- It primarily contains functions for generating AES, 3DES and RSA keys. Consequently, the generateAesKey function generates an AES-256 key through the CKM_AES_KEY_GEN mechanism. It sets various attributes for the key, such as token, private, sensitive, encrypt, decrypt, and key size. Similarly, it offers the generateDes3Key and generateRsaKeyPair functions for the generation of the 3DES and RSA key pairs, respectively.

Encryption and Decryption of Data :- The function encryptData initiates the necessary mechanism for encryption. For instance, CKM_AES_CBC_PAD for AES encryption and then encrypts the plain data in this buffer. The function decryptData initiates the very mechanism necessary for decryption, decrypts the encrypted data, and stores this in the decryptedData buffer. All the necessary PKCS#11 operations and error checks are handled by these functions.

Performance Metrics:-

Time Measurement: In this regard, time measurement will be of paramount importance to help ascertain the performance of the cryptographic operations. The code measures the elapsed time for key generation, encryption, and decryption operations by calling the clock function. This function shall return the processor time that the program consumed, which in turn shall be converted into seconds. The performance functions will call the relevant cryptographic operations and capture the start and end times to return the duration.

Memory Usage: Yet another important performance metric is memory usage. The getCurrentMemoryUsage function obtains the current memory usage of the process after calling GetProcessMemoryInfo. That function returns a PROCESS_MEMORY_INFO structure, which holds detailed information on how much memory is in use by a process, including the size of the working set, that is, memory in use, and the peak size of the working set in use. This function uses these functions to calculate the memory usage before and after cryptographic operations are performed, estimating the extra overhead in memory incurred by such operations.

CPU Usage: CPU usage shows the computational load that the cryptographic operations exert on the processor. The function getCurrentCpuUsage returns the value of CPU usage based on the time elapsed. It captures the CPU time before and after the cryptographic operation was executed and computes the difference. With this value, one gets an idea of the CPU load for the operation. This monitor detects operations that require high computational resources and hence need optimization.

Throughput: Throughput is the number of performed operations per any chosen unit of time. The calculateThroughput function measures the throughput of a given operation by executing it repeatedly for the prescribed number of iterations and calculating the total elapsed time. Usually, the throughput is given in operations per second, which clearly indicates the capacity of the system to deal with the cryptographic task. High throughput is needed in many applications of cryptography, such as very fast and efficient cryptographic processing for secure communications and data encryptions.

Main Function :- It reads the command line arguments for obtaining the slot ID and the PIN. Further, it loads the HSM library, then connects to the slot, measures the performance of the key generation, encryption, and decryption operations, and evaluates the initial and final memory and CPU usage. Afterwards, it prints the results. At the very end, it will print in hexadecimal and plain text both the plain data, the encrypted data, and the decrypted data, disconnecting from the slot and freeing the allocated resources.

IV. Experimental Results

A. Experimental Setup

The Hardware Security Module Environment is setup using SoftHSM2 and OpenSC. OpenSC consists of a set of software tools and libraries to work with cryptographic capabilities such as authentication, encryption and digital signatures. OpenSC implements the PKCS #11 API. Public Key Cryptographic Standard #11 [PCKS#11] is a platform independent API for cryptographic tokens such as HSM. We Implement C++ programs for comparing the three cryptographic algorithms AES, triple DES and RSA

B. Analysis

Table-1 contains the performance comparison of Throughput and Key Generation Time for Encryption-Decryption processes with three different Algorithms AES, 3DES and RSA. Table-2 contains the performance comparison of CPU usage and Memory Usage for Encryption-Decryption processes with three different Algorithms AES, 3DES and RSA.

Algorithm	Throughput (operations/sec)		Key Generation Time (sec)
	Encryption	Decryption	
AES	100000	50000	0.002
3DES	100000	99700.9	0.00503
RSA	7692.31	226.757	0.274

Table I: Performance Metrics of Cryptographic Algorithms

Algorithm	CPU Usage (%)	Memory Usage (Bytes)
AES	0.81551	147456
3DES	0.81806	946176
RSA	74.2949	339968

Table II: Performance Metrics of Cryptographic Algorithms

The plots for Encryption Throughput, Decryption Throughput, and the comparison between them are shown in figures- 6, 7 and 8 respectively.

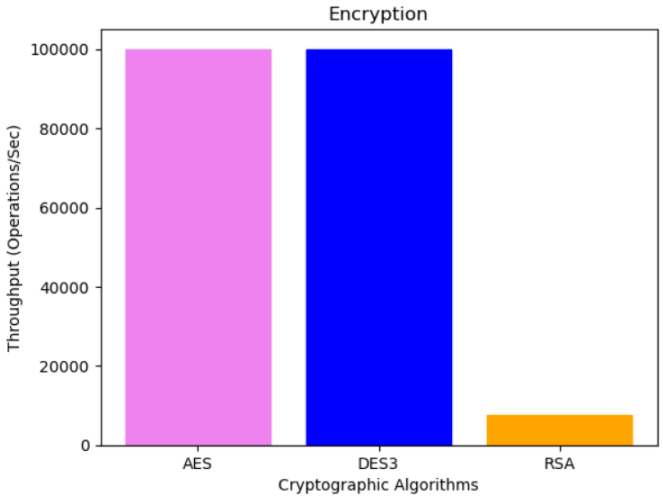

Figure 6: Encryption Throughput.

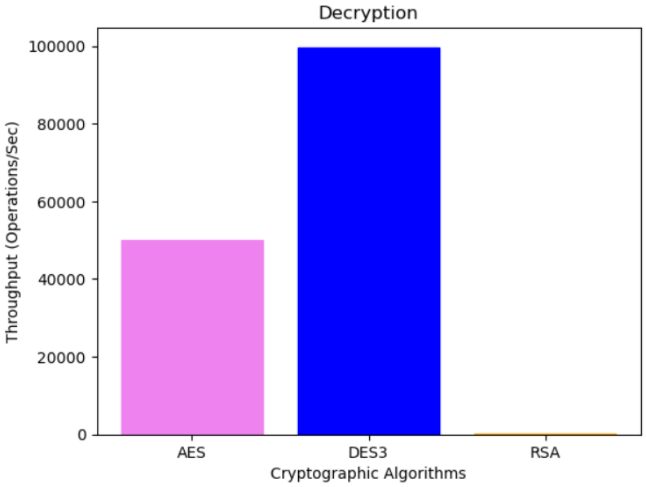

Figure 7: Decryption Throughput.

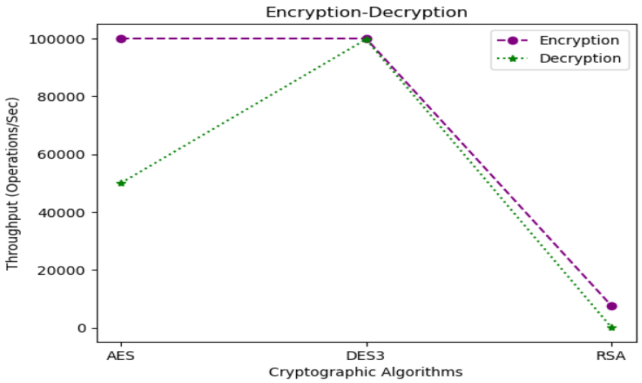

Figure 8: Comparison.

The plots for Key Generation Time, CPU Usage, and Memory Usage are shown in figures- 9, 10 and 11 respectively.

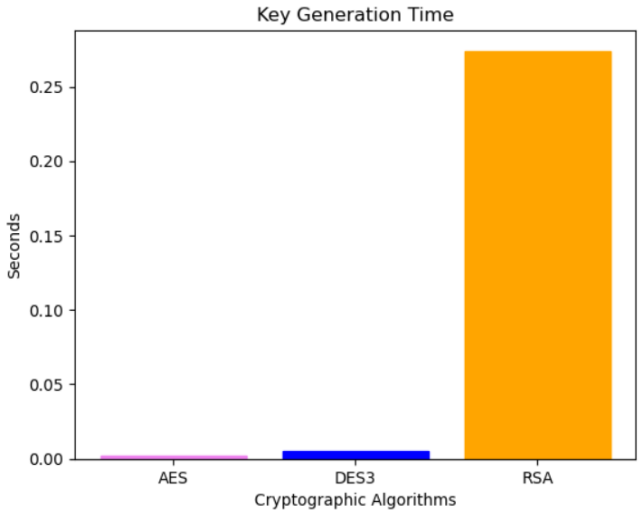

Figure 9: Key Generation Time.

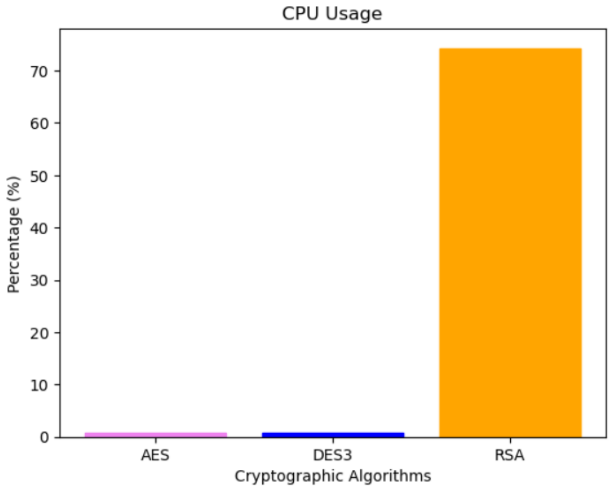

Figure 10: CPU Usage.

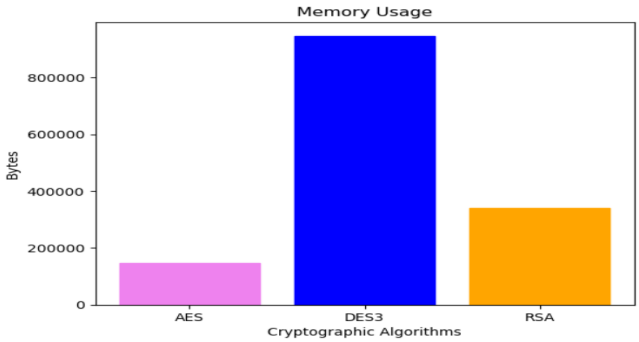

Figure 11: Memory Usage.

We infer that AES is the most efficient algorithm in terms of key generation time, throughput, CPU usage, and memory usage, making it highly suitable for high-performance and resource-constrained applications. 3DES offers good throughput but at the cost of higher memory usage and slightly higher decryption time. RSA is computationally intensive with high CPU usage and lower throughput, which makes it less suitable for high-speed encryption tasks but essential for secure key exchange due to its asymmetric nature.

C. Discussion

In this project, we utilize SoftHSM2, an open-source emulation of a physical Hardware Security Module (HSM), to perform basic cryptographic operations and evaluate the performance of three widely used algorithms: AES, 3DES and RSA. SoftHSM2 provides a controlled environment to analyze and compare these algorithms based on various performance metrics.

After implementation we find that, AES has the best rate among block ciphers because of its speed, security and flexibility, including support for key lengths of 128, 192, and 256 bits, with no known practical attacks against it when it is used correctly. Thus making it highly suitable for modern cryptographic applications.

3DES is more secure than DES, given that it is triple-encrypted. At the same time, the advantages include the compatibility it retains with DES. Despite this, it is slower and less efficient compared to AES, having a relatively small effective key length.

RSA is the most preferred asymmetric encryption algorithm which offers the advantage of high security but is slower than symmetric-key algorithms, specifically when processing large volumes because it does significant amount of mathematical computing processes and requires larger key sizes for equivalent security.

The appropriateness of each encryption algorithm ultimately depends on the specific use case:

AES is preferred for securing sensitive data such as encrypting databases and files, securing communications in VPNs and messaging platforms. It is also employed in streaming services for real-time encryption, securing wireless networks according to standards like the wi-fi Protected Access security standard (WPA2), and implementing secure cloud storage solutions.

3DES remains relevant for legacy systems and specific scenarios where backward compatibility with DES is required, such as certain financial transactions and older enterprise applications. However, due to its slower performance and reduced efficiency compared to modern algorithms, it is generally phased out.

RSA is suitable for asymmetric encryption tasks, such as secure key exchange and digital signatures. It is widely used in securing web communications through protocols like HTTPS and in establishing secure email communications. RSA is also employed in public key infrastructure (PKI) systems to manage and validate digital certificates.

V. Conclusion

In this work, we setup a Virtual Hardware Security Module environment using SoftHSM2. We implement C++ programs for comparing cryptographic algorithms [RSA, 3DES and AES] based on performance metrics such as Encryption Throughput, Decryption Throughput, Key Generation Time, Memory Usage and CPU Usage.

Through our experimental and theoretical comparison, we have demonstrated that AES outperforms the other algorithms in terms of overall efficiency, particularly in encryption and decryption throughput. 3DES, while secure, shows significantly slower performance and higher resource consumption. RSA, as an asymmetric algorithm, also shows slower performance in throughput compared to AES but is essential for secure key exchanges.

From these insights, we conclude that AES is the most optimal algorithm for environments where high performance and low resource usage are critical, but RSA remains indispensable for tasks requiring secure key exchanges and digital signatures due to its high security.

REFERENCES

[1] "What is a Hardware Security Module (HSM)? - Utimaco," utimaco.com, Apr. 05, 2022. Accessed: June. 15, 2024. [Online.] Available: https://utimaco.com/service/knowledge-base/hardware-security-modules/what-hardware-security-module-hsm

[2] "What is an HSM? — What are the benefits to using an HSM? — Encryption Consulting," Sep. 23, 2020. Accessed: June. 15, 2024. [Online.] Available: https://www.encryptionconsulting.com/education-center/what-is-an-hsm/

[3] A. Lance, "What is a Hardware Security Module? — Sidechain Security," Jun. 28, 2024. Accessed: Jun. 15, 2024. [Online.] Available: https://sidechainsecurity.com/what-is-a-hardware-security-module/

[4] W. Stallings, Cryptography and network security: principles and practice: William Stallings. Upper Saddle River, N.J.: Pearson/Prentice Hall, 2006. pp. 27-372.

[5] P. Kumar Tiwari, V. Choudhary and S. Raj Aman, "Analysis and Comparison of DES, AES, RSA Encryption Algorithms," 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India, 2022, pp. 1913-1918, doi: 10.1109/ICAC3N56670.2022.10073996 .

[6] E. Davies and C. McKenzie, "hardware security module (HSM)," SearchSecurity, Jul. 28, 2021. Accessed: Jun. 21, 2024. [Online.] Available: https://www.techtarget.com/searchsecurity/definition/hardware-security-module-HSM

[7] "What is Hardware Security Module (HSM)? — Fortinet," Fortinet. Accessed: Jun. 21, 2024. [Online.] Available:https://www.fortinet.com/resources/cyberglossary/hardware-security-module

[8] C. Paar , J. Pelzl, "The RSA Cryptosystem." in: Understanding Cryptography : A Textbook for Students and Practitioners. Berlin, Heidelberg.: Springer Science & Business Media, 2010.

[9] "Hardware security module," Wikipedia, Jul. 12, 2024.Accessed: June. 21, 2024. [Online.] Available: https://en.wikipedia.org/wiki/Hardware_security_module

[10] "PKCS#11". Available: https://cyberhashira.github.io/notes/pkcs11/pkcs-11v2-20.pdf

[11] "Difference between RSA algorithm and DSA, GeeksforGeeks". May 22, 2020. Available: https://www.geeksforgeeks.org/difference-between-rsa-algorithm-and-dsa/

[12] "Triple DES (3DES), GeeksforGeeks". Mar. 06, 2024. Available: https://www.geeksforgeeks.org/triple-des-3des/

[13] "Advanced Encryption Standard (AES), GeeksforGeeks". May 22, 2023. Available: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

[14] "Key Generation." Available: https://github.com/CyberHashira/PKCS-11-Tutorials/tree/main/samples/object_management/generating_keys

[15] "Encryption_Decryption". Available: https://github.com/CyberHashira/PKCS-11-Tutorials/tree/main/samples/crypto_operations/encryption

[16] R. Andriani, S. E. Wijayanti and F. W. Wibowo, "Comparision Of AES 128, 192 And 256 Bit Algorithm For Encryption And Description File," 2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2018, pp. 120-124, doi: 10.1109/ICITISEE.2018.8720983.

[17] A. Hamza and B. Kumar, "A Review Paper on DES, AES, RSA Encryption Standards," 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 2020, pp. 333-338, doi: 10.1109/SMART50582.2020.9336800.

[18] K. Srinivasan, A. Akash, P. Jaiganesh and M. Mahizhan, "A VLSI Perspective on Encryption Algorithm Analysis," 2024 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 2024, pp. 1-6, doi: 10.1109/IC3IoT60841.2024.10550324.

[19] P. Parikh, N. Patel, D. Patel, P. Modi and H. Kaur, "Ciphering the Modern World: A Comprehensive Analysis of DES, AES, RSA and DHKE," 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 838-842, doi: 10.23919/INDIACom61295.2024.10498330.

[20] K. Patel, "Performance analysis of AES, DES and Blowfish cryptographic algorithms on small and large data files," International journal of Information Technology, vol. 11, pp. 813–819, January 2019. doi: 10.1007/s41870-018-0271-4

[21] M. Paradesi Priyanka et al., "A Comparative Review between Modern Encryption Algorithms viz. DES, AES, and RSA," 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES), Greater Noida, India, 2022, pp. 295-300, doi: 10.1109/CISES54857.2022.9844393.

[22] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-7, doi: 10.1109/ICEngTechnol.2017.8308215.