National College of Ireland

RAE FUSION: A Hybrid Model for Security Enhancement

MSc Research Project
MSc in Cybersecurity

Ansh Ashwini Jain
Student ID: 23308320

School of Computing
National College of Ireland

Supervisor: Mr. Jawad Salahuddin

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Ansh Ashwini Jain
Student ID:	23308320
Programme:	MSc in Cybersecurity **Year:** 2025
Module:	Practicum - II
Supervisor:	Mr. Jawad Salahuddin
Submission Due Date:	11th August 2025
Project Title:	RAE FUSION: A Hybrid Model for Security Enhancement
Word Count:	6215 **Page Count**: 21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ansh Ashwini Jain
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

RAE FUSION: A HYBRID MODEL FOR SECURITY ENHANCEMENT

Ansh Ashwini Jain

23308320

Masters in Cybersecurity

National College of Ireland

ABSTRACT

This research focuses on developing a layered encryption and decryption mechanism that integrates three widely recognized cryptographic algorithms—AES, RSA, and ECIES—each known for its distinct strengths. RSA, a widely used asymmetric algorithm, provides robust key exchange; AES, a fast symmetric cipher, ensures rapid data encryption; and ECIES, known as a hybrid encryption scheme built on top of ECC which is known for its lightweight yet secure operations, adds an extra layer of protection with smaller key sizes.

A thorough literature review was conducted, revealing that while various studies have explored combinations of two of these algorithms, such approaches often suffer from specific security vulnerabilities and performance limitations. The primary goal of this work is to address those shortcomings by creating a unified technique that leverages the combined benefits of all three algorithms [1]. To address these issues, a nested hybrid approach is proposed, where AES encrypts the data, RSA encrypts the AES key, and ECIES is used to sign or further secure the process. The implementation is carried out through custom Python code, supported by CrypTool2 for visualization and AIDA64 for system-level testing.

Performance is evaluated by measuring clock speed utilization, encryption-decryption time, and resistance to cryptographic attacks. The results confirm that the proposed model maintains high security while optimizing execution time, making it a viable solution for modern secure communication systems.

Keywords: RSA, AES, ECIES, hybrid

Contents

1. Introduction 5
1.1 Research Question 5
1.2 Goal and Purpose 5
2. Literature Review 6
2.1 Introduction 6
2.2 Related Work 8
2.3 Research Gap 9
3. Research Methods 10
3.1 Introduction 10
3.2 Primary and Secondary Sources 10
3.3 Quantitative Research 10
3.4 Procedural Approach 11
3.5 Software Used 11
3.6 Metrics 12
4. Design and Implementation 13
4.1 Encryption 13
4.2 Decryption 15
5. Results and Discussion 17
5. 1 Time 17
5.2 CPU Performance 18
5.3 CPU Performance 18
5.4 Avalanche Test 19
6. Conclusions and Future Work 20
7. References 21

1. Introduction

With the rapid expansion of digital communication and data exchange, ensuring the confidentiality and integrity of information has become a top priority in cybersecurity. Traditional encryption methods, while effective, often face limitations in balancing speed, security, and computational efficiency. To address these challenges, this study proposes a hybrid encryption model that combines three widely used cryptographic algorithms: RSA, AES, and ECIES.

Individually, each cryptographic algorithm has its own set of limitations, making them less effective when used in isolation. However, when strategically combined within a hybrid encryption system, their respective weaknesses can be mitigated by the strengths of the others. Currently, there exists a noticeable gap in research focused on building an all-encompassing hybrid encryption model that unites the core advantages of RSA for robust security, AES for high-speed data processing, and ECC for lightweight and efficient key management. This study addresses that gap by proposing and analysing a layered encryption framework designed to optimize the balance between computational performance, data protection, and system resilience.

A detailed literature review highlights existing dual-algorithm approaches and their shortcomings, establishing the need for a more comprehensive solution. This study not only introduces a nested encryption-decryption mechanism but also implements and tests it using Python and CrypTool2. Performance is assessed through simulations focused on clock speed, encryption/decryption time, and overall security metrics. The results validate the model's effectiveness in achieving a high level of security while maintaining computational efficiency, making it a promising solution for modern secure communication systems.

1.1 Research Question

R.Q. How can RSA, AES, and ECC encryption techniques be integrated into a hybrid encryption model to ensure optimized security, resource efficiency, and real-time performance in large-scale IoT networks?

The question with this research is to understand whether combing the three greatest encryption techniques, AES, RSA and ECC, into a single hybrid system, serves the purpose well. The study is conducted to understand the efficiency and the security it can serve when compared with other hybrid techniques presented in data communication systems**.**

1.2 Goal and Purpose

The purpose of this research is to develop a hybrid cryptographic model combining the Advanced Encryption Standard (AES), Rivest–Shamir–Adleman (RSA), and Elliptic Curve Cryptography (ECC) to enhance data security, performance, and scalability in modern communication systems. The integration of these three algorithms is intended to leverage the strengths of each: AES for its speed and efficiency in symmetric encryption, RSA for secure key exchange using asymmetric encryption, and ECC for providing strong security with smaller key sizes, reducing computational overhead and resource consumption [2].

This hybrid model aims to overcome the limitations associated with the individual use of these algorithms. While AES is fast, it lacks secure key exchange mechanisms; RSA provides secure key transmission but is computationally intensive with large key sizes; ECC, on the other hand, offers similar levels of security to RSA with much smaller keys, making it suitable for low-power and constrained devices. By combining them, the model ensures fast data encryption, secure and efficient key exchange, and adaptability across various platforms. Following is an image of comparison between the three different types of encryption models which are Single, Dual Hybrid and Triple Hybrid.

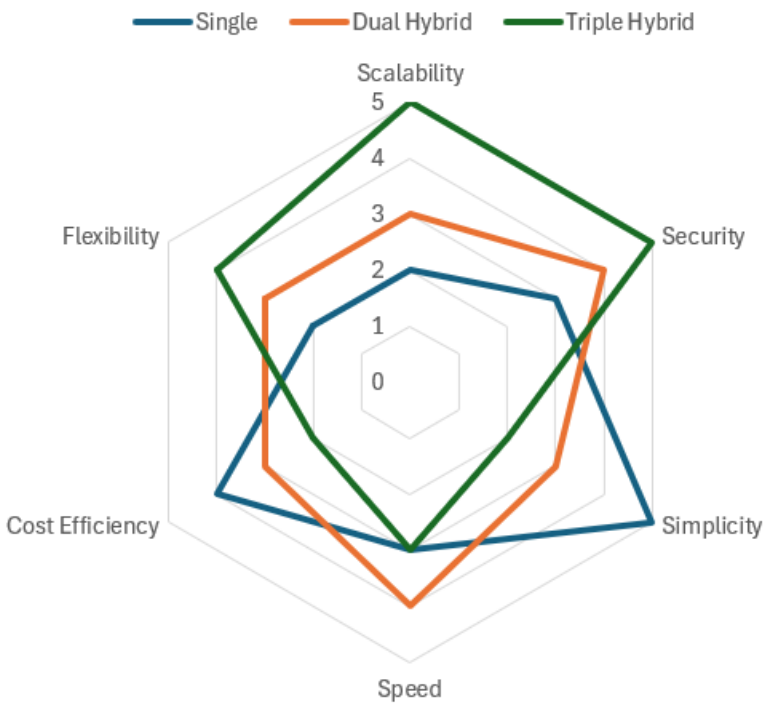

Figure 1. Comparison of the three models

The research will also evaluate the security resilience, computational performance, and practical feasibility of the hybrid model in real-world applications, making it suitable for deployment in environments such as secure communications, cloud computing, and Internet of Things (IoT) systems.

2. Literature Review

2.1 Introduction

Cryptography and encryption play a foundational role in securing digital communication, ensuring confidentiality, integrity, and authentication in an increasingly interconnected world. As data breaches and cyber threats escalate, cryptographic methods have become essential to safeguard sensitive information across sectors like finance, healthcare, and defense. According to a study by Stallings and Brown [3], modern encryption algorithms such as AES and RSA are integral to protecting data both in transit and at rest. Public key infrastructure (PKI) enables secure key exchange over untrusted networks, as outlined by Zhou et al. [4],enabling secure e-

commerce and email communication. Encryption also underpins blockchain technologies, where cryptographic hashing ensures data immutability. Furthermore, lightweight cryptography is gaining traction for securing IoT devices, which face unique constraints.

There are two different types of encryption techniques namely Symmetric and Asymmetric encryption. Both techniques work in the same logic, take a plaintext and convert into a ciphertext but in their own ways.

Symmetric encryption, which the author has used is AES (Advanced Encryption Standard), meaning the same key is used for both encryption and decryption. It encrypts data in fixed 128-bit blocks using keys of 128, 192, or 256 bits. AES operates through multiple rounds of substitution, permutation, and mixing processes to secure the data. Its speed and efficiency make it ideal for encrypting large volumes of data in real-time systems. AES is widely used in applications like VPNs, file encryption, and secure communications [5].

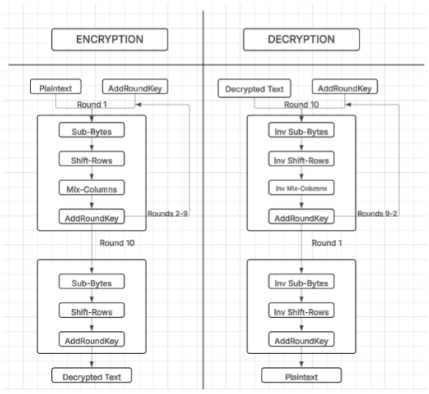

Figure 2. AES Encryption and Decryption

Asymmetric encryption, which the author selects RSA (Rivest–Shamir–Adleman) and ECIES (Elliptic Curve Integrated Encryption Scheme) meaning they use a public key for encryption and a private key for decryption.

- RSA is based on the mathematical difficulty of factoring large integers. It uses large key sizes (typically 2048 bits or more) to provide strong security but is relatively slower and computationally heavy. It's widely used in digital signatures, SSL/TLS, and secure email [6].
- ECC is based on the algebraic structure of elliptic curves over finite fields. It achieves comparable or better security than RSA with much smaller key sizes (e.g., 256-bit ECC ≈ 3072-bit RSA), making it faster and more efficient, especially for mobile and IoT devices [7].

Both are essential for secure key exchange, digital signatures, and protecting data in untrusted environments.

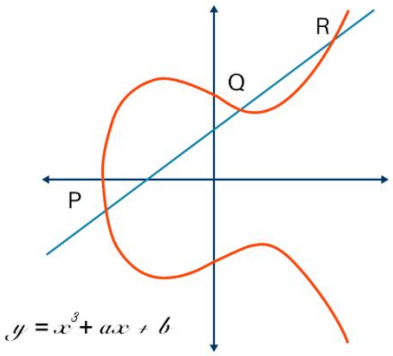

Figure 3. ECC Description

There are various encryption techniques that are present, and all are being used widely across the world. No doubt in considering the fact that any algorithm provides 100% security. Hence as we progressed through, we started to come across the concept of hybrid encryption, wherein the disadvantage of one algorithm is overcome by the advantage of other.

A hybrid encryption technique works effectively because it combines the strengths of both symmetric and asymmetric encryption methods to overcome their individual limitations. Asymmetric encryption (like RSA or ECC) is secure for key exchange but computationally slow and inefficient for encrypting large data. On the other hand, symmetric encryption (like AES) is fast and efficient for bulk data encryption but requires both parties to share a secret key securely. In a hybrid system, the asymmetric algorithm is used to encrypt the symmetric key, and the symmetric algorithm is then used to encrypt the actual data. This allows the system to benefit from the speed of symmetric encryption and the secure key distribution of asymmetric encryption, making it scalable, efficient, and secure. Hybrid encryption is widely used in protocols like SSL/TLS and PGP [8].

2.2 Related Work

Following is how the research for the said topic has been conducted till now. This is a collection of research conducted and collated through various sources present to understand what has been already done what new needs to be done.

This one is focused on understanding how the algorithms RSA and AES are incorporated and how the performance of them affects the simulation time. The evaluation is based on the time taken for both techniques of encryption and decryption with varying size of packets. The conclusion is reached that AES is faster compared to RSA [9].

The study starts with the explanation of cryptography in basic terms and then goes on to explain the main encryption ciphers. The focus then shifts to the same as authors which are RSA and AES, visualised through diagrams and tables and the contrast between them [1].

The paper suggests using RSA, ECC, and AES methods to strengthen the security of video communications. By focusing on protecting video data through chunk-based processing, it highlights the benefits of this approach and clearly explains how different keys are generated and the time required for this key generation process [10].

Here we discuss the hybrid cryptographic algorithms, namely AES-ECC-Hash, AES-DES-RSA. A slight mention of the combination that this study focuses on which is RSA-AES-ECC but not much into depth and concluded with how the security of video digital content is plausible [11].

The analysis explores how AES can be used in combination with ECC, outlining a step-by-step experimental procedure. A high-level diagram is included to illustrate the algorithm's flow. To evaluate performance, metrics like Avalanche effect, the time taken for encryption and decryption, the size of the encrypted file (in Kb), and the correlation value are considered [7].

The authors conducted an in-depth evaluation of AES and RSA, explaining each algorithm in detail with step-by-step visuals. They put both algorithms through five different tests using files ranging from 10kb up to 2mb, tracking and comparing how long encryption and decryption took. Their findings showed a clear pattern: as file size increases, it takes more time to encrypt and decrypt [5].

This rigorous study compares AES and RSA alongside other encryption methods. It details each algorithm's strengths and weaknesses, referencing their use in three separate case studies. The comparison helps identify which algorithm excels under certain circumstances and helps shape the right approach for each scenario [12].

The paper also examines the vulnerabilities of AES and ECC, describing various attack types such as brute force, Related-Subkey, and Biclique attacks. Each method of attack is explained thoroughly, along with strategies that could help mitigate these risks. It's worth noting that the study is focused on broad research, with final results not disclosed. The analysis is thorough and far-reaching [2].

The research explores combining RSA, AES, and ECC, leading to new AES-RSA and AES-ECC hybrid techniques. File sizes with different bit lengths are tested and compared. The key takeaway is that mixing any two algorithms fosters innovation and growth in cloud security [13].

2.3 Research Gap

Although there has been considerable research on encryption methods such as AES, RSA, ECC, and their hybrid combinations, much of it has primarily focused on evaluating performance-related parameters like encryption and decryption time, file size handling, and adaptability in different use cases. However, there is a noticeable lack of studies that assess the security robustness of these encryption models in conjunction with their performance. Most existing literature tends to either simulate execution times or compare individual algorithms in isolation, without thoroughly analysing their resistance to attacks. This creates a gap in understanding how well these hybrid encryption schemes perform under real-world security threats. This is the main goal of this research; to fill this gap by examining both the efficiency and security strength of hybrid encryption algorithms, offering a more comprehensive evaluation that includes vulnerability testing alongside performance metrics.

Paper/Author	Findings	Limitations
A. Hamza and B. Kumar, "A Review Paper on DES, AES, RSA Encryption Standards," in 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Dec. 2020, pp. 333–338.	AES (symmetric) is fast and well-suited for large data. RSA (asymmetric) supports secure key exchange. The paper notes that hybridization leverages the strengths of both techniques.	Focus remains largely introductory, hybrid implementation not experimentally analyzed. No empirical/test results for AES+RSA+ECC hybrid models specifically.
D. P. Mahajan and A. Sachdeva, "A Study of Encryption Algorithms AES, DES and RSA for Security," 2013.	AES consistently outperforms RSA in speed for encryption/decryption. Relevant metrics (encryption/decryption time) were rigorously measured.	Research does not include ECC nor address full AES+RSA+ECC hybridization. Limited scope to file-based speed and small data sets.
C. Manuel, "An Overview of Hybrid Cryptographic Algorithms in Information Security".	The integration of RSA-AES-ECC is plausible for digital content protection.	Analysis is conceptual—lacks in-depth technical or experimental validation. Does not detail efficiency or performance trade-offs. Practical implementation, compatibility, and overhead are not discussed.
Y. Fouzar, A. Lakhssassi, and M. Ramakrishna, "A Novel Hybrid Multikey Cryptography Technique for Video Communication," IEEE Access, vol. 11, pp. 15693–15700, 2023	Proposes a multi-key method (RSA, ECC, AES) splitting video data to enhance security. Evaluation covers multi-key generation and time needed for keys.	Increased key management complexity with multiple algorithms. Real-time feasibility for very large/latency-sensitive applications not fully addressed.
P. Kumar Tiwari, V. Choudhary, and S. Raj Aman, "Analysis and Comparison of DES, AES, RSA Encryption Algorithms," in 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Dec. 2022, pp. 1013–1018	Provides detailed methodology for testing and evaluation.	Security strength vs. performance not comparatively quantified for full hybrids. Study restricts itself to AES and RSA; ECC and full hybrid omitted.
S. Sharma and V. Chopra, "ANALYSIS OF AES ENCRYPTION WITH ECC," 2016.	AES+ECC hybrid improves avalanche effect and security for cloud/file data. Analysis includes correlation, avalanche, encrypted file size, and speed.	Limited to ECC-AES; no RSA integration, so not full triple hybrid. Practical implementation complexity (keys, compatibility) not covered in depth.
scheme combined symmetric encryption algorithm with asymmetric encryption algorithm," in *5th International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM 2023)* , Oct. 2023, pp. 484–488	Combination models (AES-RSA, AES-ECC) show improved cloud security. Hybridization helps with key distribution and efficient bulk encryption	Does not explicitly combine all three (AES+RSA+ECC) in implementation/evaluation. Hybrid complexity and potential side-channel attack resistance not assessed.

Figure 4. Research Individual Findings and Limitations

3. Research Methods

3.1 Introduction

This section outlines the methodology adopted for the study. It details the procedures followed, the environment in which the research was conducted, and the software tools required to carry it out. Additionally, it highlights the key parameters considered during the analysis, such as execution time and security aspects. The section also explains how the overall technique was developed, implemented, and evaluated throughout the research process.

3.2 Primary and Secondary Sources

Primary research for this study involved directly exploring a research question by gathering and analysing various articles and conference papers. Analysing various surveys available, developing your own algorithm, making observations or collecting information firsthand are many ways. In a nutshell, existing studies are used as a base to figure what more can be explored off and something new can be brought to life. Secondary Research on the other hand means studying materials already published. N number of journals, articles, books and online resources which are readily available to support and present the findings. Both of the sources are considered very valuable, but they serve very different purposes.

The author uses his own approach to produce results and findings of his own.

3.3 Quantitative Research

As Pritha Bhandari explains in her 2020 overview of quantitative research [14], this approach focuses on collecting and analysing numerical data. In our study, we use measurable metrics—like CPU performance, encryption/decryption speed, avalanche effect scores, and the effort

required for cryptanalysis—which makes a quantitative method ideal. By relying on these clear, quantifiable results, we can confidently assess how well our technique performed both in terms of speed and data security.

3.4 Procedural Approach

A procedural approach involves systematically planning, building, testing, and evaluating the results of an algorithm. In this study, the author began by researching the algorithm in depth, identifying the strengths and limitations of each method, and choosing a suitable platform to run the algorithm. For testing and analysis, the author used Python along with AIDA64 as the primary environment.

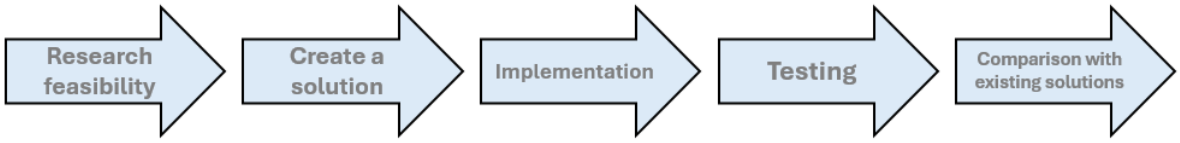

Figure 5. Procedural Approach Flowchart

The author chose AES, RSA, and ECC for the hybrid encryption algorithm due to their individual strengths and how they complement each other. AES was selected for its speed and efficiency in handling large volumes of data using symmetric encryption. RSA brings secure key exchange and digital signature capabilities, making it ideal for protecting the AES key. ECC was chosen for its strong security with smaller key sizes, which enhances performance without compromising protection. By combining these three, the author aimed to build a balanced model that ensures both high-speed processing and robust data security.

3.5 Software Used

The author chose CrypTool2 [15] because it offers a user-friendly graphical interface ideal for visualizing encryption workflows. It supports both AES and RSA with built-in modules, allowing easy configuration without requiring programming skills. The platform includes testing tools like time measurement, cryptanalysis simulations, and visualization blocks, which help in evaluating algorithm performance. Its drag-and-drop workflow setup makes it intuitive to build complex encryption pipelines. CrypTool2 is well-documented, which simplified the learning curve. It also includes educational features like real-time encryption visualization, aiding better understanding of internal operations. The ability to monitor intermediate stages of encryption and decryption made debugging and testing efficient. CrypTool2 runs on Windows and integrates smoothly with performance tools like AIDA64 Extreme [16]. It provides reproducible results, ideal for academic research and testing. Overall, its focus on cryptographic clarity and practical analysis made it a suitable choice for implementing and evaluating AES and RSA.

The author chose Python due to its flexibility and wide range of cryptography libraries that support ECC operations. Python allows direct control over ECC key generation, point arithmetic, and custom curve usage, which is essential for fine-tuned experimentation. Visual Studio Code was selected for its lightweight nature, powerful extensions, and built-in support

for Python development. The interactive environment made it easy to write, test, and debug ECC scripts efficiently. Using Python also enabled seamless integration with performance measurement libraries, aiding in tracking encryption/decryption time. It supports modular coding, making it easier to isolate and improve specific ECC functions. Open-source libraries like ecdsa and cryptography offer reliable implementations of ECC for academic research. VS Code's IntelliSense and linting features ensured fewer syntax errors and smoother coding experience. The platform also supported version control integration, which helped manage different versions of the ECC scripts. Overall, Python and VS Code offered the control, transparency, and simplicity needed for implementing and analysing ECC [17].

The author selected AIDA64 to monitor system performance during algorithm testing, particularly CPU usage and temperature. It provided real-time insights while running encryption and decryption processes, helping to measure the computational load. The tool's detailed hardware diagnostics allowed the author to assess how efficiently the system handled each method. Its clean interface and exportable data made analysis and documentation easier. Overall, AIDA64 proved valuable in evaluating the resource impact of the encryption algorithms.

The author initially explored tools like MATLAB, OPNET, and NS-3 for creating and testing the encryption model. MATLAB offered strong mathematical capabilities and visualization features, while OPNET and NS-3 provided network simulation environments that could simulate encryption in real-world data flow. However, these platforms required more complex setups and were better suited for large-scale network simulations or mathematical modelling rather than hands-on encryption workflow testing. After comparison, the author found that the current tools—CrypTool2 for AES and RSA, and Python with Visual Studio Code for ECC—offered more practical control, ease of use, and direct cryptographic functionality. These tools allowed for quicker implementation, better visualization, and easier performance analysis, making them the more effective choice for the study's goals.

Tool Name	Function	Reasoning
Cryptool2	A visual platform for designing and analysing cryptographic algorithms.	Selected for its ease of use of designing
Python with VS	Environment setup which enables customer development and execution of encryption algorithms.	Selected for the various functionalities and background in
AIDA64	Monitors and reports real-time system performance like CPU Usage and temperature.	Selected for real-time CPU usage
MATLAB	Performs complex mathematical modelling with visualization	Not selected for complexity reasons
OPNET	Simulates and analyzes the performance of network protocols and architectures.	Not selected as free software is not available
NS-3	Offers a discrete-event network simulator for research and experimentation in communication networks	Not selected due to absence of internal encryption decryption tools

Figure 6. Tools Used

3.6 Metrics

For this research, there are three main metrics that are taken into account which are Time, CPU performance and security. The evaluation is based of the RSA and AES algorithm and the author follows NIST standards while assessing the said factors.

Another test is the avalanche effect test and the key sensitivity test both conducted for their own reasons. Avalanche is used to confirm sensitivity to input changes and to ensure the output is unpredictable after changing a bit. Key sensitivity is conducted to see the reaction of encryption with slightly different keys.

CPU usage, the clock speed and the other factors in regard to the CPU were tested using the AIDA64 software. These are considered for understanding of running the algorithm.

Time was taken as a factor in consideration with the pre-understanding that running a hybrid algorithm will be taking much more time compared to singular algorithms. It's necessary and important and cannot be left out while presenting everything. In-built cryptool2 functionality has been used by the author to measure the time taken for encryption. This function calculates the time in seconds as well as milliseconds for clarity.

This research evaluates a hybrid encryption algorithm based on three key metrics: time, CPU performance, and security. Tests such as the Avalanche Effect and Key Sensitivity were used to assess encryption strength, while AIDA64 monitored CPU behavior. CrypTool2 was used to measure encryption time, acknowledging that hybrid algorithms inherently take longer than individual methods.

4. Design and Implementation

This section focuses on the encryption and the decryption process of the flow, explaining the implementation at Cryptool2 and the code.

4.1 Encryption

The Text Input Block contains the plaintext message "Test Data" which is 9 characters long and spans one line. This message serves as the input that will undergo encryption, and the output encoding is set to ASCII. In the Random Number Generators section, two Random Number blocks are connected. One of these is responsible for generating the AES Initialization Vector (IV) or key material, while the second is used for additional IV. Randomness plays a critical role in both AES-CBC mode—where a secure IV is essential—and RSA key generation, where unpredictable prime values ensure cryptographic strength.

The RSA Key Generator Block is configured to use manually entered prime numbers, labelled as p and q, to generate RSA keys. These primes are very large, with values such as *p: 1082003368681583585…* and *q: 9722539470089747217…* , ensuring strong cryptographic security. From these two primes, the RSA algorithm derives the public and private key pairs, which are essential for secure encryption and decryption in the RSA framework.

The AES Encryption Block has the following setup with the block taking the plaintext from Text Input and using the Random Number Generators for the Key and Initialization Vector. These are the same keys that are used for the decryption as well. Following are the settings on the block:

- Algorithm: Advanced Encryption Standard (AES).
- Action: Encrypt.
- Key size: 256-bit (32 bytes).
- Mode: Cipher Block Chaining (CBC).
- Padding: PKCS#7 (ensures plaintext blocks align with AES 16-byte block size).

The String Encoder Block is used to convert the raw AES ciphertext into Base64 format, making the encrypted output safe for text-based representation and transmission. The maximum output length for this encoding is set to 65,536 characters, allowing for large data

handling. Following this, the Text Output Block (AES Result) displays the resulting Base64-encoded AES ciphertext, for example: *aYF+h0VWr3vP…*This output represents the encrypted version of the original plaintext message.

The RSA Encryption Block is configured to perform encryption, most likely using the public key generated in the RSA Key Generator block. The process utilizes 12 cores, indicating that the computation is parallelized for efficiency. While an option to override block sizes is available, it remains unchecked, so the default block sizes are used. In this block, the AES key, or potentially a combination of the IV and key, is encrypted using RSA to ensure secure key transmission.

The Text Output Block (RSA Result) displays the RSA ciphertext in hexadecimal format, with output such as *38 A6 C9 20 4B A1 F5 95 D7 B3 A4 0F 6C E7 ….* This ciphertext represents the encrypted AES key, a combination of the AES key and IV, secured using the RSA algorithm. The output is block formatted, which is consistent with the standard structure of RSA-encrypted data. Hence this becomes the RSA-AES Encrypted text which will be further used by ECC to further encrypt.

The logic for ECC in python is as following:

1. Import necessary libraries

- Imports ECC operations (tinyec), secure random number generators (secrets, get_random_bytes), and SHA-256 hashing (hashlib).

2. Define encrypted data

- rsa_aes_encrypted_message: A byte string that simulates encrypted data (e.g., from RSA), which will now be encrypted again using ECC.

3. Select ECC curve

- Loads the brainpoolP256r1 elliptic curve for encryption

4. Generate ECC key pair

- Generates a random ECC private key (ecc_priv).
- Derives the corresponding public key (ecc_pub) by multiplying with the curve's generator point.

5. Define ECC encryption function which the ecc_encrypt_msg(pub_key, message_bytes) handles:

- Ephemeral key generation.
- Shared secret derivation using ECDH.
- AES key derivation from the shared secret (via SHA-256).
- AES-GCM encryption of the input message_bytes.

6. Encrypt the RSA-AES data using ECC

- Calls ecc_encrypt_msg() with the receiver's ECC public key and the RSA-AES encrypted message.
- Returns encrypted data including ciphertext, IV, tag, and ephemeral public key.

7. Output the results

- ECC private key
- Ephemeral public key (x, y coordinates)
- AES-encrypted ciphertext (in hex)
- IV and authentication tag used for AES-GCM

8. Outcome

- The RSA-AES encrypted message is now wrapped in ECC-AES hybrid encryption, enhancing confidentiality using asymmetric encryption (ECC) with forward secrecy via ephemeral keys.

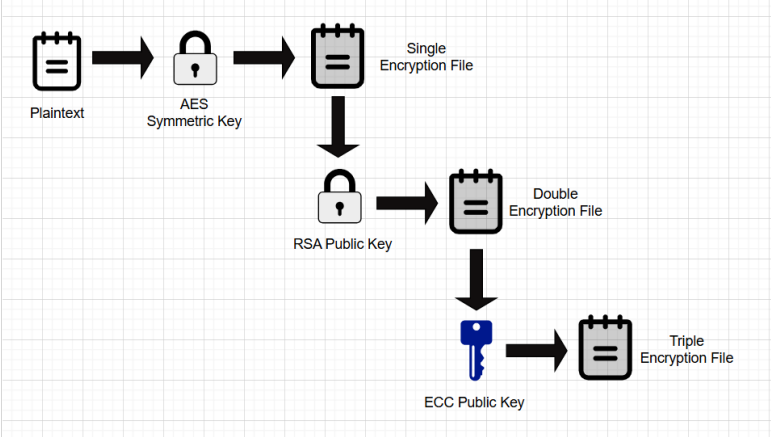

Figure 7. Encryption Phase

4.2 Decryption

Here's how the code for ECC Decryption follows as:

1. Import libraries and setup values

- Imports ECC tools, AES encryption, hashing (SHA-256), and hex handling.
- Defines x and y coordinates of the sender's ephemeral ECC public key.
- Defines the ECC curve (brainpoolP256r1) and receiver's ECC private key.

2. Provide encrypted data

- Encrypted AES data (ciphertext_hex), initialization vector (iv_hex), and authentication tag (tag_hex) are defined as hex strings.
- These are outputs from prior encryption using ECC-AES hybrid technique.

3. Convert values to correct formats

- Converts:
  - x, y into an ECC Point object.
  - Hex values (ciphertext_hex, iv_hex, tag_hex) back to binary using binascii.unhexlify().

4. Define ECC decryption function

- ecc_decrypt_msg():
    - Derives shared ECC secret using ECDH: shared_secret = ephemeral_pub_point * priv_key.
    - Converts the x-coordinate of the shared secret to bytes and hashes it with SHA-256 → derived_key.
    - Decrypts the ciphertext using AES-GCM with the derived key, IV, and tag.

5. Perform decryption

- Calls ecc_decrypt_msg() with all necessary values to decrypt the data.
- Result is the original plaintext message that was encrypted earlier.

6. Output the decrypted message

- Prints the final decrypted message a previously RSA-AES encrypted payload to verify successful decryption.

The further process for the decryption begins at the Cryptool2 with the Text Input (Hex) block, where the RSA ciphertext bytes—originally decrypted from ECC—are pasted in hexadecimal format. This input should match the expected RSA ciphertext, typically starting with values as we saw above "29 B0 5F 81 D6 D9 00 AC ... FC 04". Next, the String Decoder (Hex) block is used, where the input format is set to hexadecimal. This block converts the provided hex string into raw bytes, preparing the data for further RSA decryption operations.

The RSA (Decryption) block is configured to perform the decryption action using 12 processing cores for efficiency. The decryption uses keys sourced from the RSA Key Generator block, where the primes p, q, and the exponent were initially provided during encryption. The output from this block corresponds to the wrapped data that was originally generated during the encryption process. The output appears as a Base64-encoded string, which represents the AES-CBC ciphertext that was produced earlier during the AES encryption step. The String Encoder (Text) to Text Output block displays the Base64-encoded string, such as *aYF+h0VWr3vP...*, allowing us to easily inspect the output. This step provides a clear and readable representation of the decrypted data for verification or further use. As this message matches with our original AES encryption, we are good to proceed further.

The output of this block is decoded to raw bytes for AES decryption and the AES key used is the exact same 256-bit AES key that was originally used during encryption. In a real-world scenario, this key typically comes from RSA decryption (if the AES key was wrapped) or is retrieved securely from a key store. The IV (Initialization Vector) must also be identical to the one used during AES CBC encryption. While the IV is not considered secret, it must be accurately transmitted or stored along with the AES ciphertext. The Ciphertext input for this decryption step is obtained from the output of the Base64 decoder.

- Algorithm: AES,
- Action: Decrypt,

- Key-size: 256 bits.
- Mode: CBC,
- Padding: PKCS#7.

This output is then passed through a string encoder to be converted to text based off of ASCII values and we get our original plaintext that we have set off to encrypt.

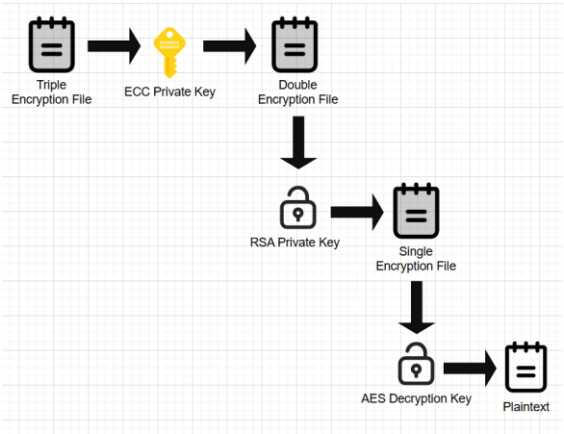

Figure 8. Decryption Phase

5. Results and Discussion

In this section we go through the conclusions we reached from the research in this section. We will test the algorithm created with three different parameters including CPU performance, encryption and decryption time and the security of the algorithm. These indicators are needed to understand how efficient and effective the hybrid model can be in the real world. The security testing included of testing the algorithm for edge cases input of the plaintext and key verifications, but for practicality purpose the focus for this study will be on Avalanche Test.

5. 1 Time

The execution time as we can in Figure 8 is of the comparison for different methods and measured in milliseconds. The whole process of encryption and decryption is considered to be called as execution time. It was tested on four different methods AES, RSA, AES+RSA, AES+RSA+ECC for three different sizes of plaintext being, 16,500 and 1024 bytes. As we focused on earlier that time is not the primary focus with this method, which can clearly be seen from the image that AES+RSA+ECC takes much more time compared to other algorithms. One thing to note here that is constant is the fact that as the byte size increases, the time taken to execute is comparatively less as the hybrid method is present to help solve complex plaintext.

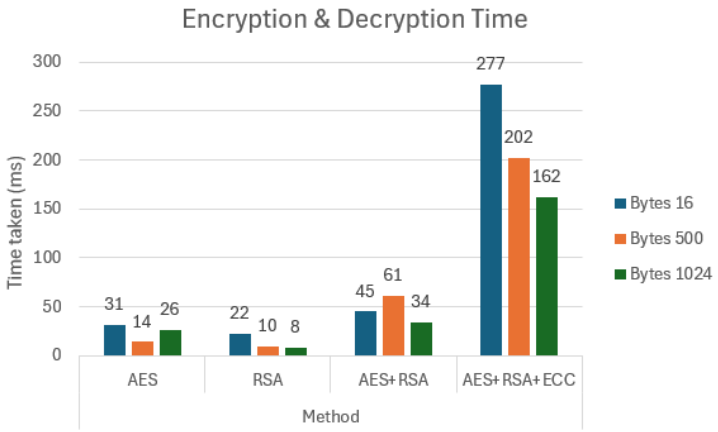

Figure 9. Time Comparison

5.2 CPU Performance

CPU clock speed as seen in Figure 9 demonstrates the rate at which the whole execution was performed. The author measured the CPU usage using the AIDA64 Extreme software which help find the clock speed. Same test size experiment that was used in above time test, was conducted for CPU and the hybrid model AES+RSA+ECC method offered fast cryptographic performance, as compared to the rest methods. This shows that the hybrid model will run securely and reasonably fast and has capabilities to handle real-time secure applications.

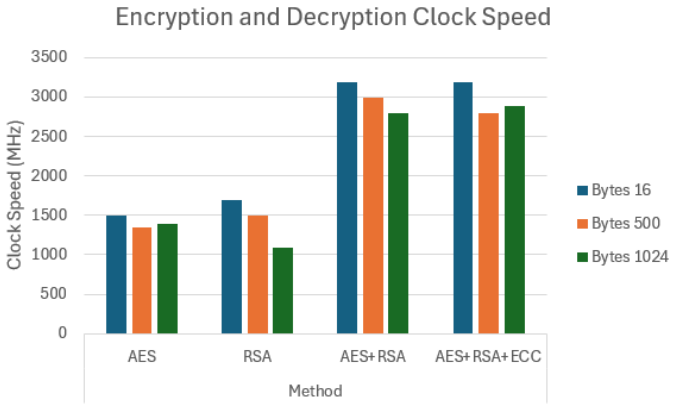

Figure 10. Clock Speed Comparison

5.3 BVA and Edge Case Testing

To ensure the reliability of the hybrid AES+RSA+ECC model, both edge cases and boundary conditions were thoroughly tested. Inputs included extremely small data (1 byte), large payloads (several kilobytes), and scenarios with malformed keys or corrupted ciphertexts. In all tested conditions, the model produced accurate and expected outputs, demonstrating robust encryption and decryption handling across diverse scenarios.

5.4 Avalanche Test

An avalanche test in cryptography evaluates whether a small change in input (like flipping a single bit) causes significant, unpredictable changes in the output, ideally flipping about 50% of the output bits. This property, called the Avalanche Effect, ensures strong diffusion and resistance to cryptanalysis. It is crucial for encryption algorithms like AES to maintain data security by obscuring input patterns [18].

Plaintext: 1001001001

AES Key: 3E 17 BA 96 AE 04 CD 3B 99 86 9E 56 F0 AD BF 3A 6F B7 4D 25 55 17 5D CC 6E 8B 09 14 57 9A B0 36

Ciphertext:
1200b857d9694da4bc7c84a3399d1094d8c142d33907b768f5dacbce0f0f17a3631237553d33c
ee1da59155edf235c0ec99d25cb8a0900ac7e6461c06cd213d02b69b50f58a8817a0aba3f0c52e
4ffc795f77b4f885078cbd434b6d0b1407c6d55bed68c6c7072ded186baf7e4f2bf975adad8e85
1af9f8fff555ed224c7cf4eabd27cc20926d6ce43fd409ab9e4346aa9a6782fe38b10cc7e6ae908d
f4d6536e45ea9762a51ed81bf80c7adccb6f448ae7cac47fb1891049e2a9d9bea2257

```
--- Encryption Output ---
Ciphertext (hex): 1200b857d9694da4bc7c84a3399d1094d8c142d33907b768f5dacbce0f0f17a3631237553d33cee1da59155edf235c0ec99d25cb8a0900ac7e6461c06cd213d02b69b50f58a8817a0aba3f0c52e4ffc795f77b4
f885078cbd434b6d0b1407c6d55bed68c6c7072ded186baf7e4f2bf975adad8e851af9f8fff555ed224c7cf4eabd27cc20926d6ce43fd409ab9e4346aa9a6782fe38b10cc7e6ae908df4d6536e45ea9762a51ed81bf80c7adccb6f448
ae7cac47fb1891049e2a9d9bea2257
Receiver Private Key: 42144518668897773556479413080647857168481212696254179334852927053172030321395
Ephemeral Public Key (x, y): (428219375488519664305688491044189031719319517528453594908749011791528443290 25, 587553899759623411532613037642433637583512237655861273094445317462595180633 49)
IV (hex): c3eef464d2074614677fa0fb
Tag (hex): 892854375a0f98248cf9eb9b888ea311
```

Figure 11. Encryption with plaintext ‘1001001001’

Changing last digit from ‘1’ to ‘0’

Plaintext: 1001001000

AES key: 3E 17 BA 96 AE 04 CD 3B 99 86 9E 56 F0 AD BF 3A 6F B7 4D 25 55 17 5D CC 6E 8B 09 14 57 9A B0 36

Ciphertext:
8fede8e64a75b4d5a2fb99d2c7b67b245cc9678d4c8f3d73284e4d466c66d35a02ddca763f61d4
5e78008acdae26392b7b17f8de64d4d6992a997f3fdae66814327a6bf9439b4b816ed53e177ea9
4fce7aba4ed3956b812957d7945e6273a2b2c7666c5b3afa2c61fadc39895cd9fc29e154418372
a298180276626bede6dca96477c4cd95787973baac08d1d782dff334c611e68218f18bb334fec2
1be04176bcd3afbc1ba75348610df8b963652d64d39a9456926f7984797ccde041758f3ff8b6

```
--- Encryption Output ---
Ciphertext (hex): 8fede8e64a75b4d5a2fb99d2c7b67b245cc9678d4c8f3d73284e4d466c66d35a02ddca763f61d45e78008acdae26392b7b17f8de64d4d6992a997f3fdae66814327a6bf9439b4b816ed53e177ea94fce7aba4ed
3956b812957d7945e6273a2b2c7666c5b3afa2c61fadc39895cd9fc29e154418372a298180276626bede6dca96477c4cd95787973baac08d1d782dff334c611e68218f18bb334fec21be04176bcd3afbc1ba75348610df8b963652d64
d39a9456926f7984797ccde041758f3ff8b6
Receiver Private Key: 11777024765665976402031542801041752325050973191376975267879592547592227844925
Ephemeral Public Key (x, y): (486108453244131868046932109345016611698830514569282447710868456128676604234 0, 278015797520089630887980582946878862883679741525921732578769531863645265761626
)
IV (hex): 12960030b8a8662911be6a61
Tag (hex): 795bbd761e341b8e7ebd2a5ee0b92890
```

Figure 12. Encryption with plaintext ‘1001001000’

6. Conclusions and Future Work

The researcher has provided the investigations to understand how a hybrid encryption and decryption model of three algorithms covering both symmetric and asymmetric encryptions can be proven useful for data communication systems. The study of the hybrid RSA, AES and ECC - namely 'RAE' method has proven itself in ways that it can enhance data confidentiality, integrity and most importantly, security. This layered approach mitigates the weaknesses of individual algorithms and presents a robust framework suitable for modern cybersecurity challenges, especially where secure communication and lightweight encryption are essential.

The evaluation demonstrated that the hybrid model could successfully encrypt and decrypt sensitive data while maintaining a balance between performance and security. AES ensured fast and reliable encryption for bulk data, RSA provided a secure mechanism for exchanging encryption keys, and ECC added an additional layer of security with smaller key sizes and lower computational overhead. Sure, the time taken for the model to run might not be as effective when compared with other single encryption methods, but considering the main goal to be security here, it definitely is effective. Testing, including performance benchmarking and avalanche analysis, validated the system’s resilience against common cryptographic attacks and its practical feasibility on standard hardware setups. This research addressed the original question of how RSA, AES, and ECC can be integrated into a hybrid model to optimize security, resource efficiency, and real-time performance in complex environments such as large-scale IoT networks. The findings suggest that the RAE method offers a promising balance by enhancing security and scalability while maintaining sufficient efficiency for practical deployment in such settings. Looking ahead, future work could focus on streamlining this hybrid process to optimize it further for real-time, high-volume applications, thereby increasing its applicability across diverse domains. Additionally, as quantum computing advances, integrating emerging quantum cryptographic techniques into this hybrid framework presents a valuable opportunity to future-proof secure communications.

7. References

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

[2] H. Tange and B. Andersen, "Attacks and countermeasures on AES and ECC," in *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, Jun. 2013, pp. 1–5. Accessed: Mar. 29, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/6618523/?arnumber=6618523

[3] W. Stallings, L. Brown, M. Bauer, and M. Howard, *Computer security: principles and practice*, 2. ed., International ed. in Always learning. Boston, Mass.: Pearson, 2012.

[4] "A Literature Review on RSA, DES and AES Encryption Algorithms," in *Emerging Trends in Engineering and Management*, 2023rd ed., Soft Computing Research Society, 2023, pp. 57–63. doi: 10.56155/978-81-955020-3-5-07.

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

[6] U. Somani, K. Lakhani, and M. Mundra, "Implementing digital signature with RSA encryption algorithm to enhance the Data Security of cloud in Cloud Computing," in *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*, Oct. 2010, pp. 211–216. doi: 10.1109/PDGC.2010.5679895.

[7] S. Sharma and V. Chopra, "ANALYSIS OF AES ENCRYPTION WITH ECC," 2016.

[8] N. Kumar, S. Kumar, A. K. Kashyap, and R. Rana, "Performance Evaluation of Cryptography Algorithms: AES, DES, RSA, and ECC," vol. 10, no. 1, 2023.

[9] D. P. Mahajan and A. Sachdeva, "A Study of Encryption Algorithms AES, DES and RSA for Security," 2013.

[10] Y. Fouzar, A. Lakhssassi, and M. Ramakrishna, "A Novel Hybrid Multikey Cryptography Technique for Video Communication," *IEEE Access*, vol. 11, pp. 15693–15700, 2023, doi: 10.1109/ACCESS.2023.3242616.

[11] C. Manuel, "An Overview of Hybrid Cryptographic Algorithms in Information Security".

[12] Prashant, Md Sohail Haque, Amrinder Kaur, and Pankaj Yadav, "Comparative Analysis of AES and RSA with Other Encryption Techniques for Secure Communication," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol*, vol. 10, no. 2, pp. 565–574, Apr. 2024, doi: 10.32628/CSEIT2410263.

[13] Y. Wang, "A data encryption scheme combined symmetric encryption algorithm with asymmetric encryption algorithm," in *5th International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM 2023)*, Oct. 2023, pp. 484–488. doi: 10.1049/icp.2023.2981.

[14] "Pritha Bhandari," Scribbr. Accessed: Aug. 07, 2025. [Online]. Available: https://www.scribbr.com/author/pritha/

[15] "About CrypTool 2 – CrypTool." Accessed: Aug. 07, 2025. [Online]. Available: https://www.cryptool.org/en/ct2/

[16] "AIDA64 Extreme | AIDA64." Accessed: Aug. 07, 2025. [Online]. Available: https://www.aida64.com/products/aida64-extreme

[17] "Visual Studio Python IDE - Python Development Tools for Windows," Visual Studio. Accessed: Aug. 07, 2025. [Online]. Available: https://visualstudio.microsoft.com/vs/features/python/

[18] "Avalanche Effect in Cryptography - GeeksforGeeks." Accessed: Aug. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/computer-networks/avalanche-effect-in-cryptography/