International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 09 | Sept 2023 www.irjet.net p-ISSN: 2395-0072

Development and performance comparison of modified RSA algorithm with other cryptographic algorithms

Deepesh Suranjandass[1], Rahul Gowlapalli[2],

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, 632014, India

[2]School of Electronics Engineering, Vellore Institute of Technology, Vellore, 632014, India

---

Abstract - *Cryptography has become an integral layer of defence to secure communications, safeguard personally identifiable information, prevent document tampering and establish trust between servers. A weak cryptographic algorithm can expose critical assets to vulnerabilities, so it is equally important to keep investing in unique and advanced solutions. This paper aims to analyse existing methods in use and provide an empirical solution that consumes fewer resources but is also equally or more reliable. After critically analysing famous cryptographic solutions we will create a new algorithm that aims to perform better than previously existing algorithm*

Key Words: **Cryptography, encryption, decryption, modified RSA, symmetric algorithm, asymmetric algorithm, hashing.**

1.INTRODUCTION

Our project's goal is to examine available encryption-decryption methods and build a new, more robust and reliable technique. Then you can compare the two. algorithm with previous algorithms to demonstrate how it is superior. AES stands for Advanced Encryption Standard. The American Encryption Standard (AES) is a symmetric square code that the US government has chosen to monitor described information. AES is utilised in both programming and equipment to encode touchy information all through the world. The Data Encryption Standard (DES) is a norm for encoding information. The National Institute of Standards (NIST) distributed DES, symmetric-key square encryption as well as innovation (NIST). DES is a Feistel Cipher execution. It utilises a 16-round construction of Feistel. The squares are 64 pieces in size. DES has a viable key length of 56 pieces, notwithstanding the way that the key length is 64 pieces. This is on the grounds that 8 of the 64 pieces of the key are not used by the encryption calculation. RSA-The concept of RSA is predicated on the fact that factoring a large number is difficult. The public is calculated by taking the product of two quintessentially large prime numbers. The same two prime numbers are also used to create a private key. As a result, if the huge number can be factored in, the private key is compromised. Accordingly, encryption strength is altogether reliant upon the key size, and as key size is multiplied or significantly increased, encryption strength increments dramatically. The length of a RSA key is normally 1024 or 2048 pieces. Instead of checking raw data to ensure that two sets of data are equal, MD5 generates a checksum for each set and compares the checksums to ensure that they are identical. Our goal is to discover the most efficient algorithm by calculating the execution time for each one separately. This is the calculation with the briefest execution time. These are a couple of the current calculations that we're assessing. We've moreover made a more secure transformation of the RSA computation. It adds an additional a level of safety by altering general society and private keys prior to involving them in the encryption and unscrambling processes. This makes the algorithm difficult to decipher

2. Related Works

2.1 Sheba Diamond Thabah, Mridupawan Sonowal, Rekib Uddin Ahmed, Prabir Saha, Fast and Area Efficient Implementation of RSA Algorithm, Procedia Computer Science, Volume 165,2019, Pages 525-531, ISSN 1877-0509.

Abstract:

The implementation strategy has been altered to achieve high-frequency operation. The key time-consuming aspects of the RSA cryptosystem, such as modulo multiplication and modulo exponentiation, are proposed in this paper. The greatest frequency attained for performing the RSA cryptosystem on 8-bits and 64-bits is 545 MHz and 298 MHz, respectively, which is faster than prior implementations.

2.2 Shakya, Aman & Karna, Nitesh. (2019). Enhancing MD5 hash algorithm using symmetric key encryption. ICCSP '19: Proceedings of the 3rd International Conference on Cryptography, Security and Privacy. 18-22. 10.1145/3309074.3309087.

Abstract:

Message genuineness and honesty are fundamental in this day and age of organizational correspondence. Cryptographic hash capacities are the main part of message honesty. Hash capacities are accessible in a wide scope of shapes and sizes. Another keyed hash work is presented and depicted in this review. No matter what the length of the information, this suggested technique creates a 128-cycle hash code. The capacity hashes a message with a key so an interloper who doesn't have the foggiest idea about the key can't translate it, and it conforms to the organization's security, validation, and trustworthiness necessities. This paper talks about the capacity plan procedure, as well as the security and specialized elements of such hash capacities.

2.3 A. Mohammed Ali and A. Kadhim Farhan, "A Novel Improvement With an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-Document," in IEEE Access, vol. 8, pp. 80290-80304, 2020, doi: 10.1109/ACCESS.2020.2989050.

Abstract:

The technique suggested in this paper improves the MD5 algorithm by incorporating a versatile different length and a highly efficient that replicates the highest security currently offered. While the logistic method was used to encrypt RNA using a new key established utilising the preliminary permutation (IP) tables used for the data encryption standard (DES) with linear-feedback lfsr (LFSR), the study suggests a number of structures to enhance the MD5 hash function.

2.4 Amorado, Ryndel & Sison, Ariel & Medina, Ruji. (2019). Enhanced Data Encryption Standard (DES) Algorithm based on Filtering and Striding Techniques. ICISS 2019: Proceedings of the 2019 2nd International Conference on Information Science and Systems. 252-256. 10.1145/3322645.3322671.

Abstract:

Quite possibly the most broadly utilized encryption strategy is the DES calculation. Be that as it may, it is helpless against animal power and cryptanalysis assaults because of its minuscule key size and simple and steady XOR activities. Instead of only executing the XOR operation, this work seeks to address the problem by employing an extension of the f- function that includes striding techniques. The new approach's key will be produced as a binary, then shifted into two halves and rotated, resulting in a subkey that will be permuted and enlarged to meet the size. In comparison to AES and other algorithms, this strategy increased the complexity of the encryption algorithm while preserving speed, performance, and efficiency.

2.5 V. S. Aparna, A. Rajan, I. Jairaj, B. Nandita, P. Madhusoodanan and A. A. S. Remya, "Implementation of AES Algorithm on Text And Image using MATLAB," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 1279-1283, doi: 10.1109/ICOEI.2019.8862703.

Abstract:

This article uses text and a picture as inputs to the AES algorithm, which produces encrypted output. The AES decryption procedure uses this encrypted output to create decrypted data. The method is implemented using MATLAB software. AES encryption and decryption are used to provide a secure communication channel for data transit. A 128-bit key is used to encrypt both the text and picture input. The complete technique is created with MATLAB software, and the simulation is then exhibited. The communication is encoded (ciphered) in such a way that it cannot be decoded by an unauthorized user. Because the decrypted output is identical to the input, there is no distortion. This highlights the AES algorithm's efficiency since it is able to retrieve the original message as the decryption result with no data leakage during the transmission. The algorithm may be used in a variety of fields, including military, finance, and intelligence.

2.6 Abid, R., Iwendi, C., Javed, A.R. et al. An optimised homomorphic CRT-RSA algorithm for secure and efficient communication. Pers Ubiquit Comput (2021). https://doi.org/10.1007/s00779-021-01607-3

Abstract:

In today's digital age, the secure and reliable interchange of information between devices is critical for any network. This data is kept on storage devices, routing devices, and via cloud communication. Cryptographic techniques are used to provide secure data transfer while also protecting the user's privacy by storing and delivering data in a certain manner. Only the intended person who has the key to the encryption may access the content. The channel should be safeguarded during data or critical transmission by employing strong encryption methods. In the past, homomorphic encryption (HE) algorithms were utilized for this purpose.

2.7 Lin C-H, Hu G-H, Chan C-Y, Yan J-J. Chaos-Based Synchronized Dynamic Keys and Their Application to Image Encryption with an Improved AES Algorithm. Applied Sciences. 2021; 11(3):1329. https://doi.org/10.3390/app11031329

Abstract:

The goal of this research was to create chaos-based synchronized dynamic keys and an enhanced chaos-based advanced encryption standard (AES) algorithm using the suggested synchronized random keys. First, a ripple control method based on sliding mode control (SMC) technology was proposed to ensure synchronization between master- slave discrete chaotic systems. The same dynamic random chaotic signals might be obtained at the transmitter and receiver in communication systems under synchronization. Then, based on chaotic synchronization, a unique modified AES cryptosystem with dynamic random keys was introduced. A static key is used in a classical AES cryptosystem, and it must be exchanged and validated in advance to be preserved safely. However, in the suggested architecture, by using chaotic system synchronization technology, the static key becomes dynamic and unpredictable, and it no longer has to be retained or broadcast via open channels. As a result, the disadvantage of key storage might be avoided, and encryption security may be increased. Finally, the newly created chaos-based AES (CAES) algorithm was used to build a unique picture encryption technique. Through simulation tests, the statistical analysis, histogram, information entropy, and correlation indices were produced and studied in order to illustrate the capacity and improvement of the proposed CAES cryptosystem.

3. Proposed Work and Methodology

3.1 Modules Description

3.1.1 Home Page :

The Home Page will consist of a simple landing page wherein the user will be allowed to choose amongst AES,DES,MD5,MRSA, and RSA Options for testing out the Algorithms with all the comparison parameters.

3.1.2 Algorithms Testing Module:

Each of the proposed 5 algorithms will have an individual page, wherein the user can enter a message, key and additional information based on the algorithm and then on submission of the options the Algorithm will be run in the background and the results will be displayed. The user has an option to encrypt, decrypt or both based on the algorithm

3.1.3 Virtualisation of Results :

The user will be able to virtualise all the results of the 5 algorithms based on the various comparison parameters mentioned earlier. Each algorithm will produce individual results for each of the 7 comparison parameters.

3.2 Methodology

The diagram below indicates the methodology of the project, wherein we will be building a web-based application. On reaching the Landing Page, the user will be able to initially choose amongst the 4 algorithms(MRSA is a variation of RSA, hence only 1 option for that(RSA)). On choosing the Algorithm, the user will be able to choose whether to Encrypt, Decrypt or do Both, and on clicking the submit button, they will be able to see the visualised results of the Analysis of that particular algorithm based on the Comparison Parameters.

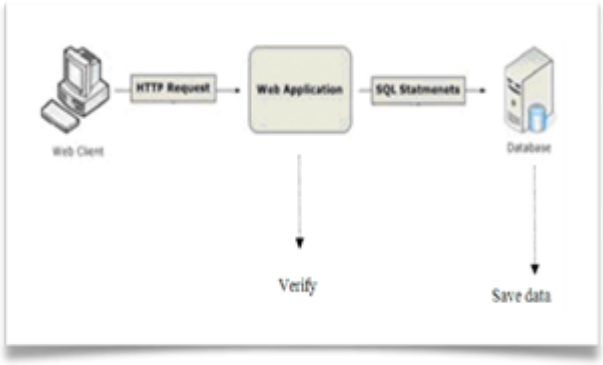

Fig 1: Web Application Structure

3.2.1 Modified Algorithm Explanation

The RSA (Rivest–Shamir–Adleman) algorithm is a widely used public-key cryptosystem for secure communication and data encryption. It relies on the mathematical properties of large prime numbers for its security. A modified RSA algorithm refers to a variant of the traditional RSA algorithm with specific changes or improvements to enhance its security, efficiency, or functionality. Here's a detailed explanation of a modified RSA algorithm:

3.2.1.1 Key Generation

In the modified RSA algorithm, key generation follows similar steps as the traditional RSA algorithm:

- Step 1: Select Two Large Primes (p and q): Choose two large, distinct prime numbers, p and q. These primes should be randomly generated to ensure security.
- Step 2: Calculate n: Compute n as the product of p and q, i.e., n = p * q. This is the modulus used for both encryption and decryption.
- Step 3: Calculate φ(n): Calculate φ(n), Euler's totient function of n, which is the count of positive integers less than n that are coprime to n. For two distinct primes p and q, φ(n) = (p-1)(q-1).
- Step 4: Select Encryption Key (e): Choose an encryption key, e, such that 1 < e < φ(n) and e is coprime to φ(n). Common choices include 3, 17, or any other small prime number.
- Step 5: Calculate Decryption Key (d): Compute the modular multiplicative inverse of e modulo φ(n). In other words, find d such that (d * e) % φ(n) = 1. This is the private decryption key.

3.2.1.2 Encryption

To encrypt a plaintext message (M), the modified RSA algorithm uses the recipient's public key (n, e). Compute the ciphertext (C) as C = M^e % n, where ^ denotes exponentiation and % represents the modulo operation. The recipient can decrypt this ciphertext using their private key.

3.2.1.3 Decryption

To decrypt the ciphertext (C), the recipient uses their private key (d). Compute the plaintext message (M) as M = C^d % n.

3.2.1.4 Security Enhancements

Padding Schemes: Padding the plaintext before encryption (e.g., PKCS#1 or OAEP padding) helps prevent certain attacks, like padding oracle attacks.
Key Management: Securely managing key pairs, including key storage, rotation, and revocation, is crucial for overall system security.
Randomness: Ensure that the prime numbers (p and q) and key generation parameters are generated with high entropy to resist attacks based on predictable inputs.
Key Length: Use longer key lengths (e.g., 2048 bits or 4096 bits) to resist brute-force attacks.
Secure Hardware: Implement the algorithm on secure hardware to protect against physical attacks.
Key Exchange: RSA can be combined with other cryptographic techniques, like Diffie-Hellman key exchange, to provide secure key exchange protocols.
Monitoring and Logging: Implement monitoring and logging mechanisms to detect and respond to any unusual activities related to RSA key usage.

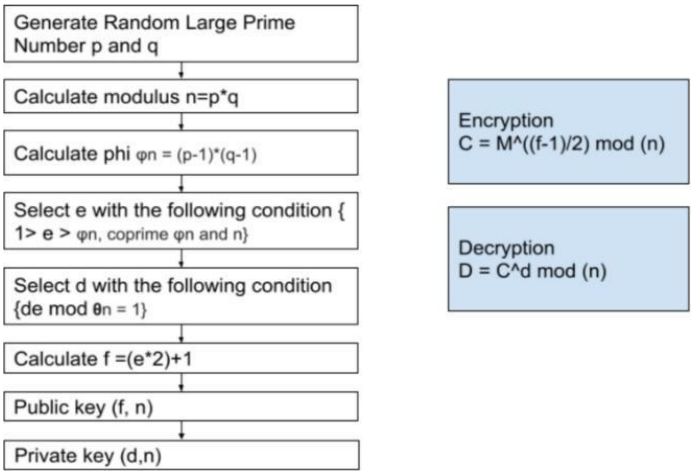

Fig 2: Modified RSA Algorithm

4. Pseudo Code for the various algorithms

4.1 Modified RSA Algorithm

```
# MRSA.py

from math import sqrt, gcd

from itertools import count, islice

class MRSA:

  @staticmethod

  def is_prime(n):

      return n > 1 and all(n % i for i in islice(count(2), int(sqrt(n) - 1)))


  @staticmethod

  def is_relatively_prime(a, b):

    return gcd(a, b) == 1

  @staticmethod

  def q_inv(q, p):

    x = 0

    # Do while xq%p is not equal to 1:

    while x * q % p != 1:

      # Increment x by one:

      x += 1

    return x

  def pow_es(self, base, power):

    # Raise to power by using exponentiation by squaring

    # If power is equal to 1...

    if power == 1:

      # return base value (stop recursion cycle)

      return base

    # If power is even...

    if power % 2 == 0:

      # call the same function with squared base and half power

      return self.pow_es(base * base, power / 2)

    # If power is odd...

    else:

      # multiply base by the value of the same function with

      # squared base and half of (power - 1) as arguments

      return base * self.pow_es(base * base, (power - 1))


  def chinese_remainder(self, c, p, q, d):

    dp = (d - 1) / 2 % (p - 1)

    dq = (d - 1) / 2 % (q - 1)

    qinv = self.q_inv(q, p)

    m1 = self.pow_es(c, dp) % p

    m2 = self.pow_es(c, dq) % q

    h = qinv * (m1 - m2) % p

    return m2 + h * q

  def chinese_remainder2(self, c, p, q, d):

    dp = d % (p - 1)

    dq = d % (q - 1)

    qinv = self.q_inv(q, p)

    m1 = self.pow_es(c, dp) % p

    m2 = self.pow_es(c, dq) % q

    h = qinv * (m1 - m2) % p

    return m2 + h * q

  def do_mrsa(self, message, p, q, e, action):

    if self.is_prime(p) and self.is_prime(q):

      if self.is_relatively_prime(e, (p - 1) * (q - 1)):
```

```
        f = (e * 2) + 1

        if action == 'encrypt':

         return {'cipher':
self.chinese_remainder(message, p, q, f), 'f': f}

        elif action == 'decrypt':

          d = self.q_inv(e, (p - 1) * (q - 1))

                                    return {'message':
self.chinese_remainder2(message, p, q, d), 'D': d, 'f': f}

        elif action == 'both':

          d = self.q_inv(e, (p - 1) * (q - 1))

          cipher = self.chinese_remainder(message, p, q, f)

                          return {'cipher': cipher, 'message':
self.chinese_remainder2(cipher, p, q, d), 'd': d, 'f': f}

        else:

                    raise ValueError('Non-available action
selected')

      else:

        raise ValueError(f'{e} is not relatively prime to {p}
and {q}')

    else:

        raise ValueError('{0} and {1} have to be prime
numbers!'.format(p, q))

def test_mrsa():

  text = 1314

  e = 343

  p = 397

  q = 401

  mrsa = MRSA()

  print(rsa.do_mrsa(text, p, q, e, 'both'))
```

Fig 3: Web Application

5. Results and Discussion

5.1 AES Algorithm

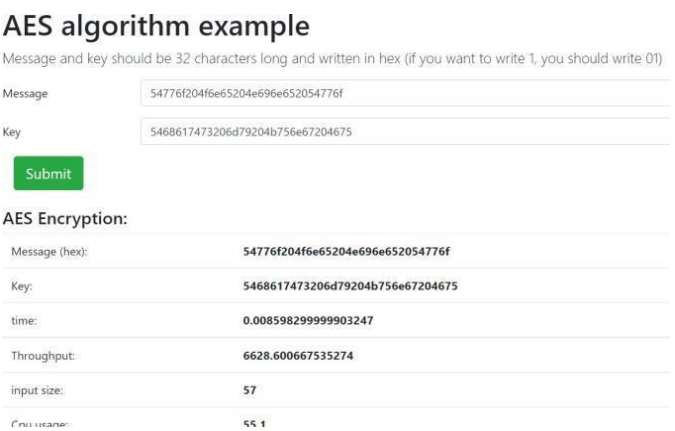

Fig 4: AES Algorithm Input Function

Round	Round Key	Sub Bytes	Shift Rows	Mix Columns	Applied Round Key
0	54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75	-	-	-	00 3C 6E 47 1F 4E 22 74 0E 08 1B 31 54 59 0B 1A
1	E2 32 FC F1 91 12 91 88 B1 59 E4 E6 D6 79 A2 93	63 EB 9F A0 C0 2F 93 92 AB 30 AF C7 20 CB 2B A2	63 EB 9F A0 2F 93 92 C0 AF C7 AB 30 A2 20 CB 2B	BA 84 E8 1B 75 A4 8D 40 F4 8D 06 7D 7A 32 0E 5D	58 15 59 CD 47 B6 D4 39 08 1C E2 DF 88 BA E8 CE
2	56 08 20 07 C7 1A B1 8F 76 43 55 69 A0 3A F7 FA	6A 59 CB BD A0 4E 48 12 30 9C 98 9E 3D F4 9B 8B	6A 59 CB BD 4E 48 12 A0 98 9E 30 9C 8B 3D F4 9B	15 C9 7F 9D CE 4D 4B C2 89 71 BE 88 65 47 97 CD	43 0E 09 3D C6 57 08 F8 A9 C0 EB 7F 62 CB FE 37
3	D2 60 0D E7 15 7A BC 68 63 39 E9 01 C3 03 1E FB	1A AB 01 27 B4 5B 30 41 D3 BA E9 D2 AA E8 BB 9A	1A AB 01 27 5B 30 41 B4 E9 D2 D3 BA 9A AA E8 BB	AA 65 FA 88 16 0C 05 3A 3D C1 DE 2A B3 4B 5A 0A	78 70 99 4B 76 76 3C 39 30 7D 37 34 54 23 5B F1
4	A1 12 02 C9 B4 68 BE A1 D7 51 57 A0 14 52 49 5B	BC 51 EE B3 38 3B EB 12 04 FF 9A 18 20 26 39 A1	BC 51 EE B3 3B EB 12 38 9A 18 04 FF A1 20 26 39	10 BC D3 F3 D8 94 E0 E0 53 EA 9E 25 24 40 73 7B	B1 08 04 E7 CA FC B1 B2 51 54 C9 6C ED E1 D3 20

Fig 5: AES Algorithm Output Function

5.1.1 Input Data Size

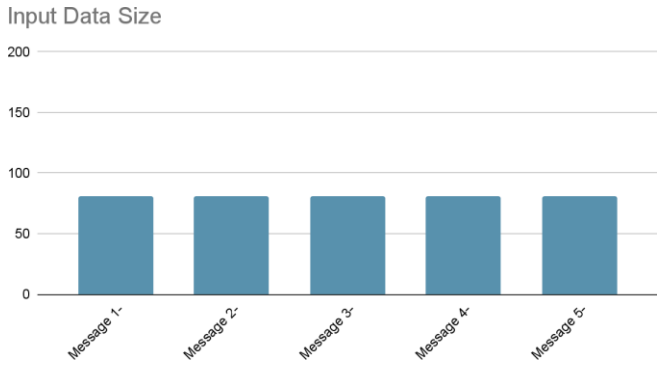

Fig 6: AES Algorithm Input Data Size

Taking identical data input sizes across all experiments is crucial to ensure the minimisation of experimental bias. Maintaining consistent data input sizes not only helps in achieving reliable and comparable results but also enhances the overall validity and reproducibility of the experimental outcomes.

5.1.2 Time

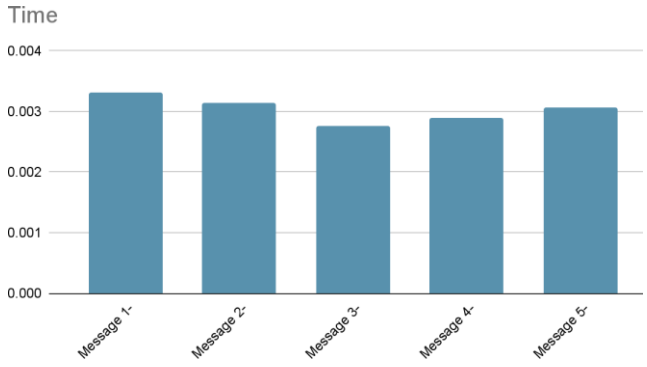

Fig 7: AES Algorithm Time Taken

Ensuring an insignificant difference in the computational time required to process identical data input sizes is a fundamental aspect of evaluating the performance of the AES (Advanced Encryption Standard) algorithm. When computational times remain consistent across experiments with identical input sizes, it signifies stability and reliability in the algorithm's execution. This uniformity in execution time not only aids in drawing meaningful comparisons between different AES configurations but also underscores the algorithm's robustness and efficiency.

5.1.3 Throughput

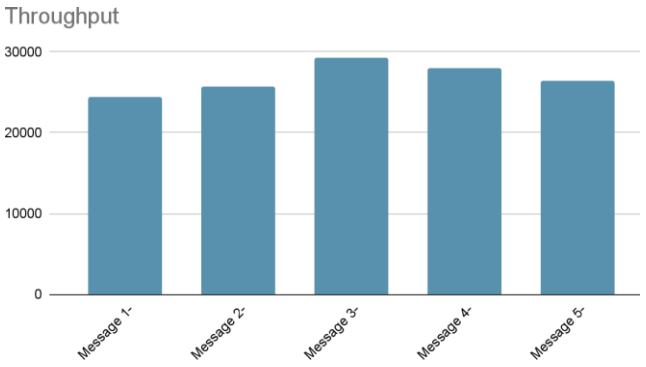

Fig 8: AES Algorithm Throughput

The variation in throughput observed in the AES (Advanced Encryption Standard) algorithm when applied to identical data input sizes is not insignificant. Analyzing the throughput under these conditions reveals that there are discernible differences in how the algorithm performs with consistent input sizes. This variability in throughput metrics highlights the importance of conducting comprehensive performance evaluations and may suggest potential areas for optimization or further investigation to achieve more consistent results.

5.1.4 CPU Consumption

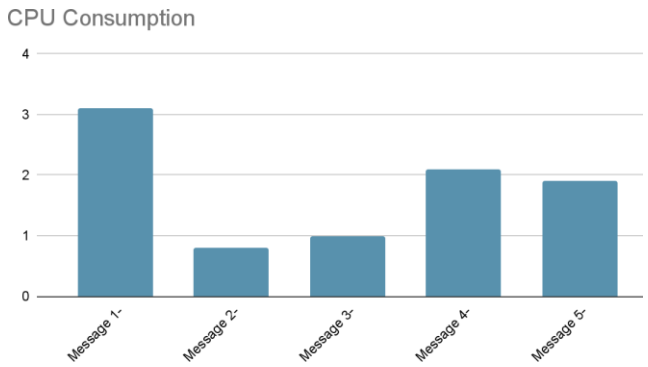

Fig 9: AES CPU Consumption

The DES (Data Encryption Standard) algorithm exhibits a significant and, at times, drastic variation in CPU consumption when subjected to identical data input sizes. This fluctuation in CPU usage underscores the sensitivity of the algorithm's computational demands to the input data, which can lead to non-trivial deviations in resource utilization. Such fluctuations highlight the need for thorough performance analysis and may warrant further investigation into the algorithm's efficiency and resource management strategies.

5.2 DES Algorithm

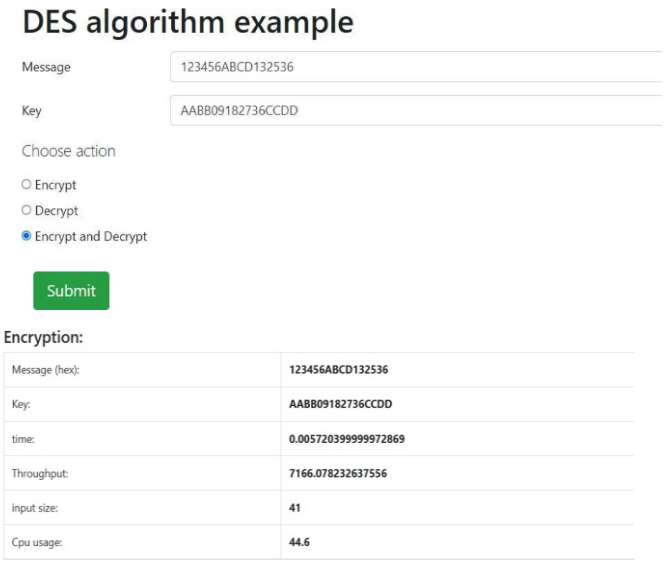

Fig 10: DES Input Function

Round	Block #1 Left	Right	Key
0	19BA9212	CF26B472	181C5D75C66D
1	CF26B472	BD2DD2AB	3330C5D9A36D
2	BD2DD2AB	387CCDAA	251B8BC717D0
3	387CCDAA	22A5963B	99C31397C91F
4	22A5963B	FF3C485F	C2C1E96A4BF3
5	FF3C485F	6CA6CB20	6D5560AF7CA5
6	6CA6CB20	10AF9D37	027657088 5BF
7	10AF9D37	308BEE97	84B84473DCCC
8	308BEE97	A9FC20A3	34F822F0C66D
9	A9FC20A3	2E8F9C65	708AD2DDB3C0
10	2E8F9C65	A15A4B87	C1948E87475E
11	A15A4B87	236779C2	69A629FEC913
12	[illegible]	[illegible]	[illegible]
13	B8089591	4A1210F6	06EDA4ACF5B5
14	4A1210F6	5A78E394	4568581ABCCE
15	5A78E394	18CA18AD	194CD072DE8C
16	14A7D678	18CA18AD	-
Cipher (after final permutation):		123456ABCD132536	

Fig 11: DES Output Function

5.2.1 Input Data Size

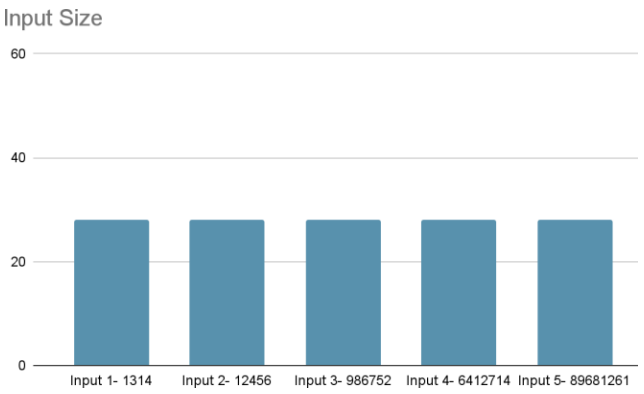

Fig 12: DES Input Data Size

Utilizing identical data input sizes across all experimental scenarios is a fundamental practice aimed at mitigating the potential sources of experimental bias. By maintaining uniform data input sizes throughout the experimentation process, researchers seek to achieve the highest level of experimental integrity and reliability. This consistency in data input sizes not only promotes fair and unbiased comparisons but also contributes to the overall robustness and validity of the experimental outcomes, ensuring that observed effects are more likely to be attributed to the manipulated variables rather than disparities in data sizes.

5.2.2 Time

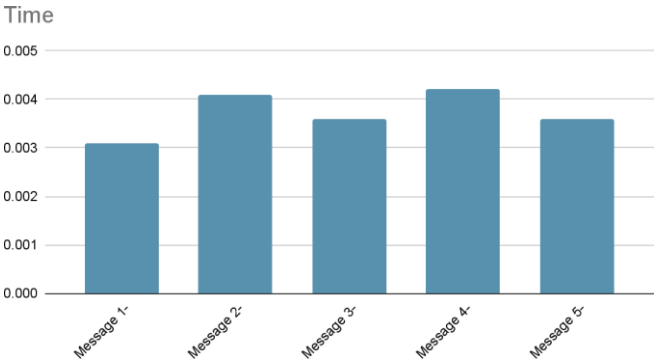

Fig 13: DES Time

The variation in time taken to compute by DES algorithm on identical data input sizes is not insignificant.

5.2.3 Throughput

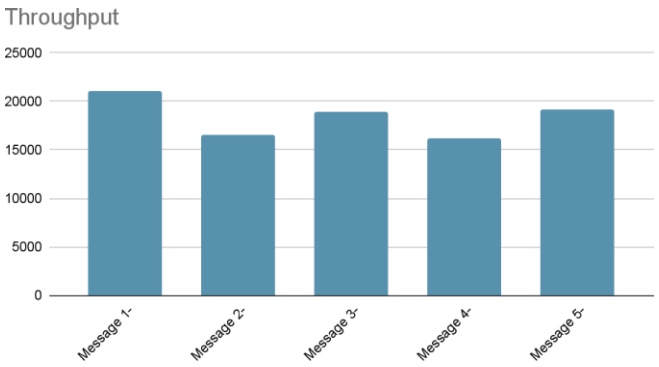

Fig 14: DES Throughput

The variation in throughput taken to compute by DES algorithm on identical data input sizes is not insignificant.

5.2.4 CPU Consumption

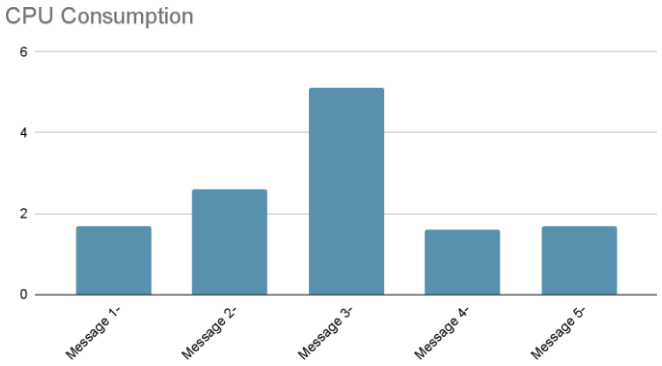

Fig 15: DES CPU Consumption

Drastic variation in CPU consumption shown by DES algorithm over identical data input sizes.

5.3 RSA Algorithm

Fig 16: RSA Input and Output Function

5.3.1 Input Data Size

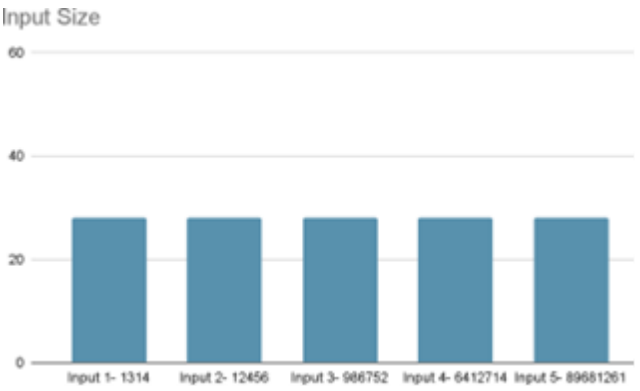

Fig 17: RSA Input Data Size

Ensuring uniform data input sizes across all experiments is imperative to minimize experimental bias effectively. The use of identical data input sizes serves as a critical control factor in research, significantly reducing the potential for experimental bias. Consistency in data input sizes not only promotes the generation of reliable and

unbiased results but also enhances the overall validity and reproducibility of the experimental findings. By adhering to a standardized input size, researchers can confidently compare and draw meaningful conclusions from their experiments, bolstering the integrity of their research outcomes.

5.3.2 Time

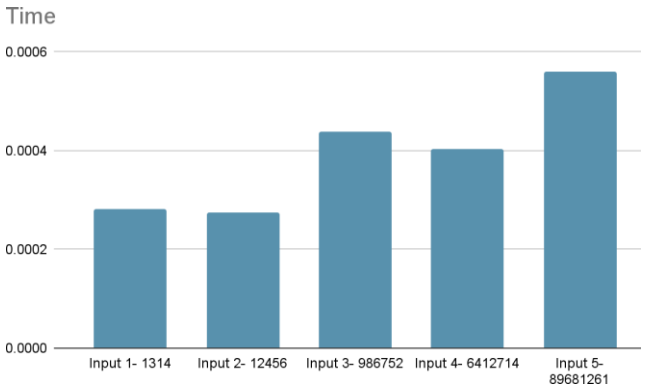

Fig 18: RSA Time

The variation in time taken to compute by RSA algorithm on identical data input sizes is significant.

5.3.3 Throughput

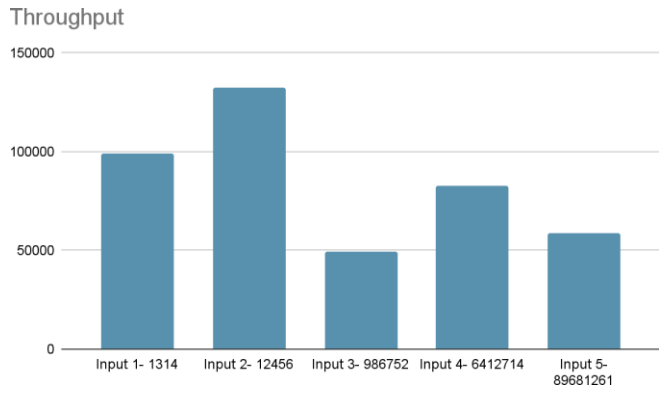

Fig 19: RSA Throughput

The variation in Throughput taken to compute by RSA algorithm on identical data input sizes is significant.

5.3.4 CPU Consumption

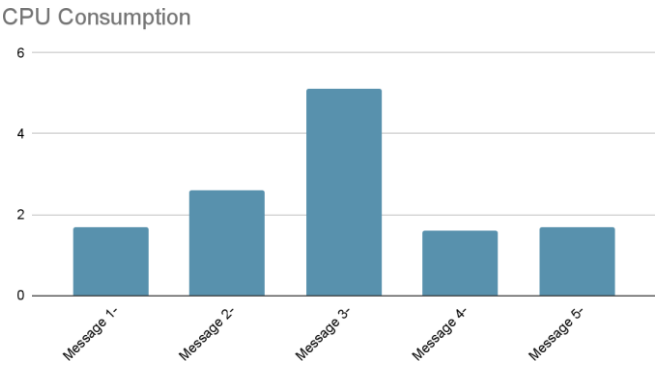

Fig 20: RSA CPU Consumption

The RSA (Rivest–Shamir–Adleman) algorithm exhibits a highly noticeable and substantial difference in CPU consumption when applied to identical input sizes. This significant variation in CPU resource utilization highlights the algorithm's sensitivity to input data, leading to pronounced discrepancies in computational demands. The magnitude of this disparity underscores the importance of comprehensive performance analysis and raises questions about the algorithm's efficiency and resource management strategies, necessitating further investigation to better understand and optimize its computational behavior.

5.4 MD5 Algorithm

Fig 21: MD5 Input and Output Functions

5.4.1 Input Data Size

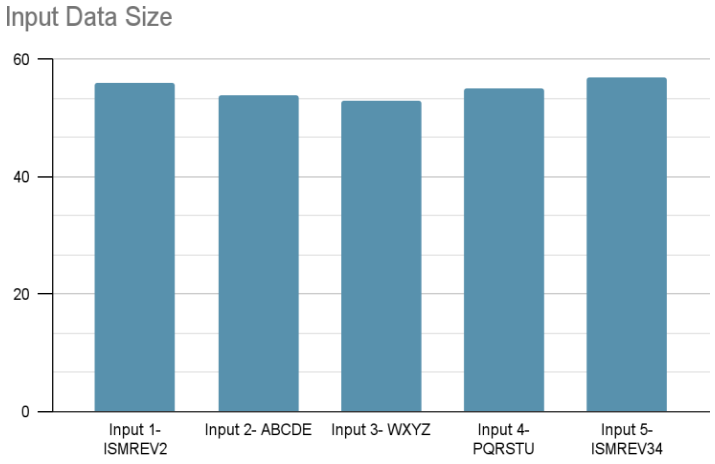

Fig 22: MD5 Input Data Size

The input data sizes are almost identical which helps to reduce experimental bias

5.4.2 Time

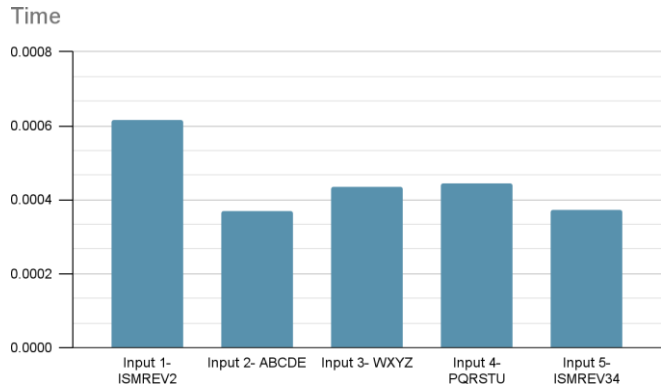

Fig 23: MD5 Time

The variation in time taken to compute by MD5 algorithm on identical data input sizes is significant.

5.4.3 Throughput

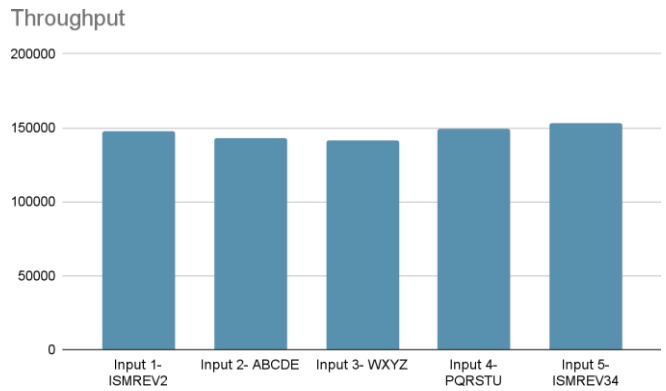

Fig 24: MD5 Throughput

Similar throughput is provided by MD5 algorithm over identical input data sizes

5.4.4 CPU Consumption

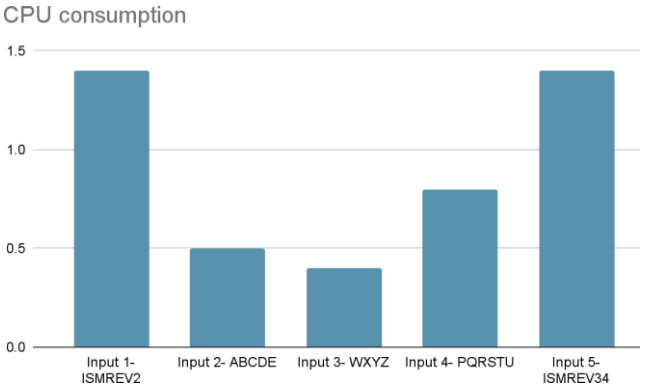

Fig 25: MD5 CPU Consumption

There is a drastic difference between the CPU consumption of identical input data sizes with the MD5 algorithm.

5.5 Modified RSA Algorithm

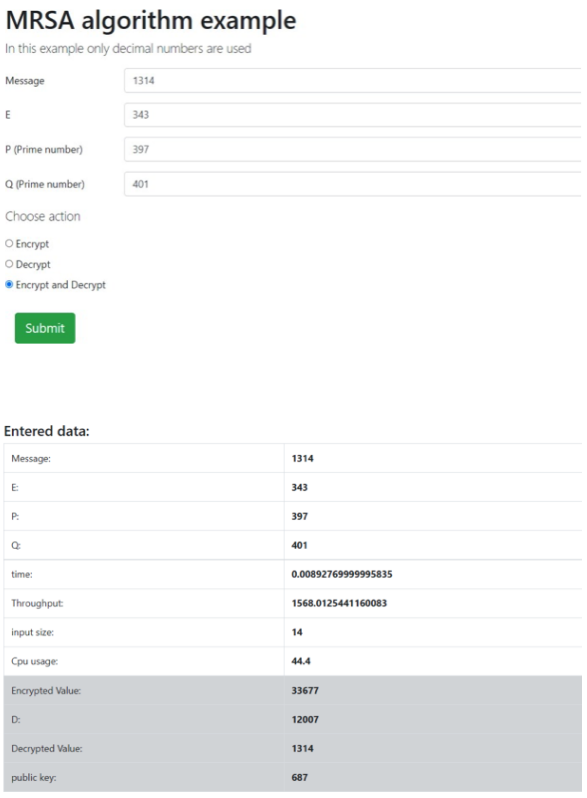

Fig 26: Modified RSA Algorithm Input and Output Function

5.5.1 Input Data Size

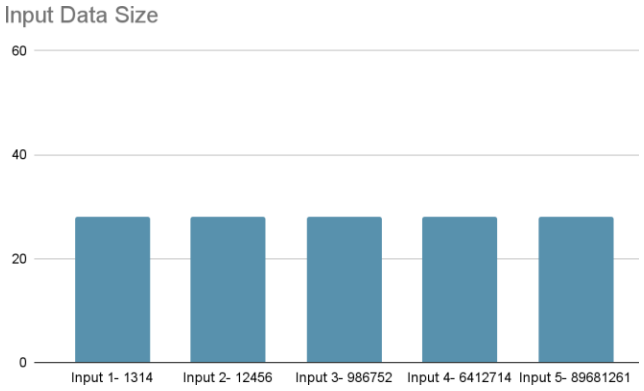

Fig 27: MRSA Input Data Size

Identical input sizes taken to get least experimental bias

5.5.2 Time

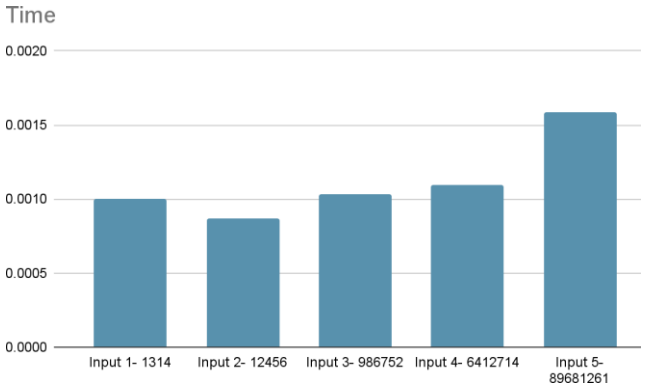

Fig 28: MRSA Time

The variation in time taken by MRSA algorithm on identical data input sizes is not insignificant.

5.5.3 Throughput

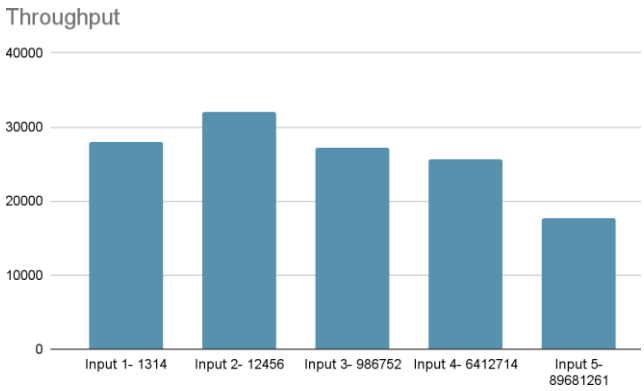

Fig 29: MRSA Throughput

The variation in throughput given by MRSA algorithm on identical data input sizes is not insignificant.

5.5.4 CPU Consumption

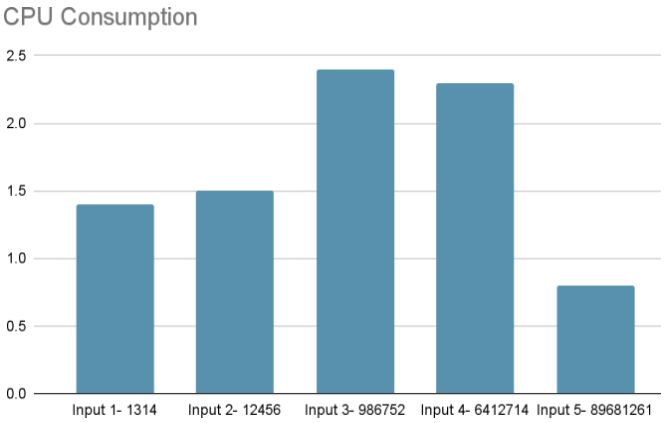

Fig 30: MRSA CPU Consumption

Drastic variation in CPU consumption shown by MRSA over identical input data sizes.

5.6 Discussion

Features	DES	AES	RSA	Modified RSA
Key Used	Same key is used for encryption and decryption purposes	Same key is used for encryption and decryption purposes	Different keys are used for encryption and decryption purposes	Different keys are used for encryption and decryption purposes
Scalability	It is scalable, due to varying key and block size	It is scalable, due to varying key and block size	No Scalability is feasible	We can modify the integer to ensure scalability
Avalanche Effect	Not affected as such	Highly affected	Highly affected	Highly affected

Power Consumption	Low	High	High	High
Time	High	Low	Very High	High
Throughput	Very High	Low	Low	Low
Confidentiality	High	High	Low	High

Table 1: Comparison of Results

6. CONCLUSIONS

With data leaks and personal chats of people happening all the time, security has become a highly vital part of modern- day technology. The primitive form of the RSA algorithm has been shown to be a good and efficient security algorithm. It does, however, have some disadvantages. The public key and private key are both susceptible, and if someone obtains a person's private key, that person's whole data is at risk. As a result, in this project, we presented a method for making the RSA algorithm more secure and overcoming its flaws, so that even if a person's private key is released, a hacker will not be able to access private information. Because of this element of our project, it is extremely important in the field of data security.

REFERENCE

[1] Sheba Diamond Thabah, Mridupawan Sonowal, Rekib Uddin Ahmed, Prabir Saha,Fast and Area Efficient Implementation of RSA Algorithm,Procedia Computer Science, Volume 165,2019, Pages 525- 531, ISSN 1877-0509, doi: 10.1016/j.procs.2020.01.024

[2] Shakya, Aman & Karna, Nitesh. (2019). Enhancing MD5 hash algorithm using symmetric key encryption. ICCSP '19: Proceedings of the 3rd International Conference on Cryptography, Security and Privacy. 18-22. doi: 10.1145/3309074.3309087.

[3] A. Mohammed Ali and A. Kadhim Farhan, "A Novel Improvement With an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-Document," in IEEE Access, vol. 8, pp. 80290-80304, 2020, doi: 10.1109/ACCESS.2020.2989050.

[4] Amorado, Ryndel & Sison, Ariel & Medina, Ruji. (2019). Enhanced Data Encryption Standard (DES) Algorithm based on Filtering and Striding Techniques. ICISS 2019: Proceedings of the 2019 2nd International Conference on Information Science and Systems. 252-256. doi: 10.1145/3322645.3322671.

[5] V. S. Aparna, A. Rajan, I. Jairaj, B. Nandita, P. Madhusoodanan and A. A. S. Remya, "Implementation of AES Algorithm on Text And Image using MATLAB," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 1279-1283, doi: 10.1109/ICOEI.2019.8862703.

[6] Abid, R., Iwendi, C., Javed, A.R. et al. An optimised homomorphic CRT-RSA algorithm for secure and efficient communication. Pers Ubiquit Comput (2021). https://doi.org/10.1007/s00779-021-01607-3

[7] Lin C-H, Hu G-H, Chan C-Y, Yan J-J. Chaos-Based Synchronized Dynamic Keys and Their Application to Image Encryption with an Improved AES Algorithm. Applied Sciences. 2021; 11(3):1329. https://doi.org/10.3390/app11031329

[8] Li, Y., HeLu, X., Li, M., Sun, Y., Wang, L. (2019). Implementation of MD5 Collision Attack in Program. In: Sun, X., Pan, Z., Bertino, E. (eds) Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science(), vol 11632. Springer, Cham. https://doi.org/10.1007/978-3-030-24274-9_54