Enhancing the Security in Group Message Communication Using Multi-Party Key Distribution Technique

Bilas Haldar[1], Rohit Sinha[2] and Pranam Paul[3,*]

[1]*Department of Computer Science and Engineering, The Neotia University, Sarisha, Diamond Harbour Road, South 24 Parganas – 743368, West Bengal, India*
[2]*AI Labs Academy Pvt. Ltd., Kolkata – 700091, West Bengal, India*
[3]*Dean – Academics and Professor, Department of Computer Science and Engineering, Global Institute of Engineering and Management, Palpara More, NH-34, Krishnanagar, Nadia, West Bengal 741102, India*
E-mail: bilasphd2020@gmail.com; rohit123sinha456@gmail.com; pranam.paul@gmail.com
[*]*Corresponding Author*

Received 07 December 2023; Accepted 20 July 2024

Abstract

Authentication and authorization techniques are essential for any kind of secure transaction over the Internet. It enables and empowers mechanisms to implement highly secure systems such as digital financial transition systems, hospitality management systems, and defence organizations. Most of the work on these techniques has been done based on single-user authorization. It is less secure than the multi-user authorization technique. Multi-party settings require the authorization of the majority user pool, the work is still in its nascent stage. Hence the presented work includes a multi-user authentication technique based on the approach of arithmetic greatest common divisor. This method aims to significantly enhance the security and reliability of the authorization process. Additionally, the proposed work also

Journal of Mobile Multimedia, Vol. 20_5, 961–996.
doi: 10.13052/jmm1550-4646.2051

presented innovative encryption and decryption techniques for the security of data transformation using a modified N prime RSA algorithm. Furthermore, this work also discussed the application of the proposed methodology in the hospitality industry to ensure the security of customer data. The results of this work indicate that the suggested techniques enhance the security of data transformation and improve the time efficiency of key generation, encryption, and decryption techniques.

Keywords: Key generation, encryption, decryption, authentication, N prime RSA, security analysis, hospitality industry.

1 Introduction

The concept of securing data predates the invention of the computer itself. The key idea behind this data security is simple, to represent the data in a form that doesn't reveal the information contained in the data. This is done to protect the data from the eyes of unauthorized and unintended recipients. This simple art of hiding data is what is known as encryption and the reverse is known as decryption. Together these two concepts form the domain of cryptography. Data Security is not only about encrypting or decrypting messages and files. A key concept is authentication and authorization. Not only encrypting data during transit is important, but also authenticating the recipient and validating their authorization. While the general growth of the cryptographic literature involves better encryption and decryption schemes, very few shed light on key generation and validating methods.

However, in recent years more and more focus is being given to key distribution techniques using quantum computing algorithms [1]. Even though a quantum computer now is more than just a theoretical concept and it is physically feasible via programmable superconducting processors [2], it's far from commercially feasibility. Thus, in this paper, we propose a novel key distribution method using classical techniques. The proposed method is designed as such that their implementation in commodity hardware including edge devices is computationally less expensive and feasible. The majority of the work involves RSA or modified RSA [3]. It focuses particularly on resisting or overcoming attacks on information systems [4–8]. Our work focuses on the novelty key distribution method which is designed Modified N prime RSA algorithm.

In this work, it has been proposed a novel key distribution technique in conjunction with the implementation of a modified RSA algorithm with the help of the "N" prime numbers. The key distribution algorithm is subdivided into two parts; key generations and key validation. The key generation step generates N key pairs for a pool of users. In the validation step if the number of users authenticating with their respective key pair is more than the minimum number of users required for that pool of users, then only the validation succeeds.

2 Literature Review

Md. Wasim et al. [8] proposed an algorithm that encrypts on the binary file, it can be used to encrypt any sort of data, including text and multimedia. R. Ghosh [9] introduced an adapted version of the RSA cryptographic algorithm, which operates as a public key encryption system. This unique variant incorporates four prime numbers, employing two public keys for encryption purposes and a single private key for decryption. N. Kumar et al. presented a method that used N prime number and bit stuffing to convert message plaintext to ciphertext. They used N prime number because it is difficult to change and bit stuffing provides an extra layer of encryption, ensuring maximum security and efficiency for data sent over the network [10]. Wen-Kai Yu et al. [11] outlined a multi-party interactive Cryptographic Key Distribution (CKD) protocol designed for use on public networks. Their approach involves an intermediary performing joint authentication based on fragment synthesis. Using a four-particle GHZ state, C. Wang et al. [12] proposed a robust, centralized multi-party quantum key distribution protocol. S. Kumar et al. [13] presented an innovative encryption technique combining Elliptic Curve Cryptography (ECC) with the Advanced Encryption Standard (AES). This hybrid approach aims to enhance data confidentiality while maintaining computational efficiency, making it particularly suitable for securing data in cloud storage systems. In the paper [14], Z. Ashraf et al. proposed a lightweight algorithm for symmetric key exchange to be used by smart devices with limited processing power. V. Lovic [15] author mentioned transmission loss in optical fibers is the fundamental problem in implementing quantum key distribution across long distances and at high communications rates, around nine out of ten photons are lost for every 50 km of fiber traveled. S. Kleis et al. [16] pointed out that the local oscillator is

positioned at the receiver site, and laser phase noise is one of the fundamental restrictions in continuous-variable quantum key distribution. Ghost imaging is a concept by Yi Kang et al. [17] presented as a technique for secure transmission and multiparty key distribution with mutual authentication.

The work done by Y. Ramadhan et al. [18] explored a dual-layer security mechanism that leverages the RSA algorithm for cryptography and the Low Bit Encoding (LBE) algorithm for steganography. S.P. Jadhav [19] examined various cryptographic schemes focused on reducing communication and computational costs for resource-constrained devices in IoT. J. W. Zhang et al. [20] introduced an enhanced multiparty quantum private comparison technique based on quantum homomorphic encryption. In the work done by S. Bhasin et al. [21], the primary focus of the work was to strengthen the post-quantum key encapsulation mechanism against side-channel attacks focusing mainly on the comparison operation of the Fujisaki-Okamoto transform. Wen-Kai Yu et al. [22] cite introduced a multi-party interactive cryptographic key distribution (CKD) protocol designed for public networks. Their novel method entails disassembling a rapid response (QR) code image into several fragment patterns (FPs) and adding unique watermarks for better security. F. O. Mojisola et al. [23] presented an encryption algorithm with a 1024-bit key length that enhances both the confidentiality and integrity of data. Furthermore, they enhanced RSA's security by implementing a bit insertion algorithm. A hybrid data security technique that combines conventional RSA encryption with Gaussian interpolation algorithms was proposed by J. K. Dawson et al. [24]. RSA is used for encryption and decryption in subsequent levels, while Gaussian first forward interpolation is used to encrypt ASCII values.

F. Zhao et al. [25] presented a novel double RSA technique to create a synthetic modulus that increases the security of the RSA algorithm. In a recent work by H. Nejatollahi et al. [26] a survey of the trends in lattice-based cryptographic methods. The work particularly focuses on challenges, the need of adoption, and fundamental proposals of using lattice in cryptography and their mapping schemes on commodity and special purpose hardware. M. Gayathri et al. [27] developed a comprehensive two-layer security model. The first layer involves Intrusion Detection, which aims to identify and mitigate unauthorized access attempts. The second layer, Biometric Multimodal Authentication, enhances user verification by combining multiple biometric traits, thus providing a robust authentication mechanism. R. Omollo et al. [28] explored the factorization of up to 12-digit semi-primes using Wolfram/Alpha, a powerful mathematical software that facilitates the

Table 1 Limitation of previous work

Author	Year	Method	Used Algorithm	Limitation
F. O. Mojisola et al. [23]	2022	Single user	Bit insertion algorithmusing1024-bitkeylength	The expense of execution time.
J. K. Dawson [24]	2022	Single user	A hybrid data security technique that combines conventional RSA encryption with Gaussian interpolation algorithms	An increase in file size results in a longer execution time.
Wen-Kai Yu et al. [22]	2022	Multi user	Presented method entails disassembling a rapid response (QR)code image into several fragment patterns(FPs)	It concentrates the wrong points more in or around the water mark embedding region when an attack occurs.
Z. Ashraf et al. [14]	2023	Single user	Proposed a lightweight algorithm for symmetric key exchange	Less secure methods and longer synchronization times
Yi Kang et al. [17]	2023	Multi user	Presented as a technique for secure transmission and multiparty key distribution with mutual authentication.	Only wireless key distribution is possible in this non-quantum field.
J. W. Zhang et al. [20]	2023	Multi user	Introduced an enhanced multiparty quantum private comparison technique based on quantum homomorphic encryption	Performing quantum computations with multiple parties in an unsecure manner.

exploration of polynomials. Jin et al. [29] proposed a three-dimensional cube algorithm that can be used to provide resilience against attacks in a post-quantum computing environment. The work done by J. Shi et al. [30] introduced a cryptographic scheme based on continuous variable quantum neural networks for designing quantum cryptographic algorithms. According to [31] when two monic polynomials composed of integer coefficients yield square-free outcomes, it is established that all positive divisors of these outcomes can be expressed as the greatest common divisor of the two polynomial values at a specific integer value.

S. Nasreen et al. [32] developed a robust cloud forensic investigation system utilizing the Distributed Key Cipher Policy Elliptic Curve Cryptography (DK-CP-ECC) algorithm. S. Deshpand et al. [33] presented the first Field Programmable Gate Array (FPGA)-based hardware design for computing the multiplicative inverse using the recently published fast constant-time Greatest Common Divisor (GCD) method. D. M. Ghadi et al. [34] implemented an innovative input-based key generation algorithm. This algorithm employs two distinct keys for encrypting and decrypting each character, significantly increasing the complexity for potential attackers J. Liu et al. [35] presented a fresh key-recovery strategy by resolving a linear equation. This work mentions the peel-off sophisticated encryption and decryption operations. S. Basu et al. [36]. Introduced three and five-input symmetric functions based on nano crossbar circuits and demonstrated how they might be used in cryptography. A thorough investigation of flexible architectures for implementing cryptographic algorithms is presented by M. Rashid et al. [37], which enables researchers and designers in the field to choose the best design strategy for a given algorithm and/or application based on their needs. A reconfigurable crypto coprocessor was proposed by X. Shang et al. [38], that uses its own reconfigurable features rather than adding additional resources to support numerous cryptography algorithms.

3 Proposed Methodology

After extensive literature survey, it has been found that most of the algorithms described the message encryption and decryption techniques. Every algorithm has some limitations of authentication, execution time, and security. In this work, it has been proposed a novel method of multi-party authentication using the arithmetical greatest common divisor technique and proposed a novel encryption method for the distribution of the authentication keys. Our approach is to particularly focus on using hard problems from the domains of classical and modern mathematical theories that specifically depend on classical computational techniques for implementations.

The concept of key distribution in the proposed method is based on the property of that the GCD of N prime numbers is 1 i.e. $GCD(k, i) = 1$, $\forall$k, i $\in$ n combined with the method of mimicking the information transfer in an artificial neural where the factor of the precise timing of spike (spike of signal) or sequence of spikes is modified with the cumulative value of them.

In the setting of a multi-party authentication scheme, each user of the Multi Party Pool (MPP) owns a unique key. For a successful authorization, the number of parties authorizing has to be greater than the threshold. In a pool of *K* parties the authorization *authorise* $\{k_1, k_2, \ldots, k_i\}$ succeeds when $i \geq$ *threshold*.

In Section 3.1 the generation of keys for the distribution is being discussed, along with its significance. The keys have been generated during the encryption process mentioned in Section 3.2, while using minimum numbers of well defined threshold number of keys are being used to perfectly decrypt the message.

3.1 Key Generation for Distribution

In the multi-party authentication scheme, the unique key for each user that is being generated is done using the Key Generation step. In Public Key Cryptography (PKC) for a single user, a pair of keys exists public key (key_{public}) and private key ($key_{private}$). The encryption of data is done using the public key and the decryption is done by the user using the private key.

$$Encrypt(D, key_{public}) = C \tag{1}$$

where *D* is the plain text or message that needs to be encrypted, *C* is the cipher text or the encrypted message and *Encrypt()* is the encryption algorithm.

When the encrypted message needs to be decrypted, the private key of the user is used.

$$Decrypt(C, key_{private}) = D \tag{2}$$

where *Decrypt()* is the decryption algorithm. For authentication using this method is done using the private key $key_{private}$ of the user. Each user using the PKC scheme has a unique pair of key.

In the proposed method, a pair of keys is generated for the user. However, the main difference of our proposed method from the public key cryptography technique is that one value of each pair is unique to the user while, the other value for a pool of user is common in our method. For a transaction T_i originating from the pool of user P_i to be successful the number of users authorising it must be greater than or equal to the threshold of the pool P_i.

Let's assume, there are *K* users in a pool P_i i.e. $\{U_i, U_2, \ldots, U_K\} \in P_i$. Each, user U_i has a pair of key (key_{pool}, key_{user}). For users in a particular pool, the key_{pool} is same but the key_{user} is unique i.e.

$$key_{pool}(U_i) = key_{pool}(U_j), \quad where\ U_i, U_j \in K \tag{3}$$

and

$$key_{user}(U_i) \neq key_{user}(U_j), \quad where,\ U_i \neq U_j \quad and \quad U_i, U_j \in K \tag{4}$$

The two Equations (3) and (4) together define the key structure in the proposed authentication scheme:

1. Shared Key_{pool}: All users in the same pool P_i share the same key_{pool}. This means the key_{pool} serves as a common key that can be used to identify or verify membership in the pool.
2. Unique Key_{user}: Each user in the pool has a unique key_{user}. This ensures that each user can be individually identified and authenticated within the pool.

These properties help in distinguishing between users and authenticating them based on their unique keys while maintaining a common reference key within the same pool.

For generating the key pairs for K users, K prime numbers are chosen $\{p_1, p_2, \ldots, p_K\}$ and a random number n_{random} is chosen. The random number is chosen between the range (range1, range2). The key_{user} is generated following the Equations (5) and (6) is used for generating the key_{pool}

$$key_{user}(i) = p_i \times n_{random} \tag{5}$$

and

$$key_{pool} = n_{random} \times threshold \tag{6}$$

where $key_{user}(i)$ for the i^{th} user is generated by multiplying the i^{th} prime number p_i with the random number n_{random}. This ensures that each user U_i has a unique key_{user} since each prime number p_i is unique.

The key_{pool} is generated by multiplying the random number n_{random} with a constant value called the threshold. This ensures that the key_{pool} is the same for all users in the pool.

The *threshold* is the minimum number of users required for authorisation of the pool transaction. Once the keys are generated, they are encrypted as proposed in Section 3.2 and sent to the users. Once the users receives their key and decrypts their key as per Section 3.2, they input their keys for authentication. The authentication of the keys are done as the method proposed in Section 3.3.

Unlike PKC, the key pair for each user are generated every time any kind of transaction is initiated and the lifetime of the key pairs are that of the lifetime of the transaction. Once the transaction is over, the respective key pairs of the users are discarded.

3.2 Implementation of Key During Encryption and Decryption

The key pair is usually used for authentication and authorization of a transaction. By transaction, it is generally referred to as the set of actions that is to be taken once the generated key is being transferred to the user and the user inputs are being authenticated [8]. The proposed method follows the method in Section 3.2.1 for generating the keys to be used in the cryptography methods in successive steps. Section 3.2.2 explains the encryption method while Section 3.2.3 lays down the decryption technique.

3.2.1 Key generation for encryption and decryption

- **Step 1:** The key generation for cryptography methods begins by select n large prime numbers. The prime numbers $pi \in$ ***PRIME*** where $i \in n$ must be random and independent of each other.
- **Step 2:** An intermediate value of N is calculated as per Equation (7)

$$N = \prod_{i=1}^{n} pi \tag{7}$$

  This means N is the product of all the selected prime numbers pi. The symbol Π denotes the product over the set of primes.
- **Step 3:** Using the N computed we compute the Euler totient function $\phi(N)$ using the following Equation (8) where q are the prime factors of n

$$\Phi(n) = n \times \prod_{q \,||\, n} \left(1 - \frac{1}{q}\right) \tag{8}$$

  Since, N is the product of all prime numbers we can reduce the computation by doing Equation (9)

$$\Phi(N) = \prod_{i=1}^{n} (pi - 1) \tag{9}$$

  This simplification is possible because the prime factors *pi* are unique and the formula leverages their unique properties.
- **Step 4:** Now, an integer e is selected such that $log(N) < e < \phi(N)$ and e and ϕ are co-prime. This will act as the public key.
- **Step 5:** Another integer d is selected such that $log(N) < d < \phi(N)$ and $e \times d \; mod \; \phi(N) = 1$. This means d is the modular multiplicative inverse of e concerning $\phi(N)$. This will act as the private key.

- **Step 6:** Another integer *kd* is selected such that $log(N) < d < \phi(key_{user}(U_i))$ and $key_{user}(U_i) \times kd \ mod \ \phi(N) = 1$. This ensures that *kd* is the modular multiplicative inverse of $key_{user}(U_i)$ modulo $\phi(N)$. This *kd* will act as the decryption key for the user.

This method ensures that messages can be securely encrypted and decrypted using the generated keys, leveraging the properties of large prime numbers, Euler totient, and modular arithmetic for security.

3.2.2 Encryption

In the proposed research, the encryption of keys is crucial for ensuring secure communication. The encryption is done as Equation (10).

$$c = m^{e \times key_{user}(U_i)} \ mod \ N \tag{10}$$

Here's a detailed explanation of this expression:

1. Message (m): The message m is the plaintext data that needs to be encrypted. This could be any data that a user wishes to securely transmit. Here, the message is the generated key pairs.
2. Exponent (e): The exponent e is part of the encryption key.
3. Unique User Key ($Key_{user}(U_i)$): The term $Key_{user}(U_i)$ refers to the unique key assigned to user U_i. This key is unique for each user and is generated as part of the key generation process.
4. Modulus (N): The modulus N is another critical part of the encryption key, typically derived from the product of two large prime numbers. It defines the size of the number space for the encryption operation and is a component of the public key.

Algorithm for Encryption

- **Step 1:** Convert each character of the plaintext message into its corresponding ASCII value and then convert the ASCII value of each character to 8-bit binary and store it in a fileb1.
- **Step 2:** Select the block size from the user say it bs.
- **Step 3:** Read the selected block size bits from fileb1 and convert the selected block size bits into an equivalent decimal number and store them in a list. A set of decimal numbers will be generated as per the block size.
- **Step 4:** Used the encryption key that is generated in section 3.2.1.

- **Step 5:** Calculate no. of bit required to represent intermediate modified value in binary format say as SZ using the equation $ceil(log_2(N))$.
- **Step 6:** Read decimal values from the list sequentially and encrypt the message using the Equation (10). Convert the intermediate output into SZ number of binary bits and store it in another file such as *fileb2*.
- **Step 7:** Count the number of bits in *fileb2*. If the number of bits is not divisible by 8, then include the dummy bit at the end otherwise no need to include the dummy bit at the end of the file.
- **Step 8:** Select sequentially 8-bit from the *fileb2* then convert it into equivalent decimal numbers. Then generate the symbols from the decimal numbers and store them in fileb3.
- **Step 9:** Sent ciphertext message (*fileb3*) to the receiver.

After the original message or file is being encrypted it is sent to the user along with the key pair over a secure channel. On receiving the message and the key pair the decryption and authentication of the keys are done. The encryption technique is implemented in Python programming language. Some lines of codes of the encryption function are given below.

```
def encrypt_step_8(self):
    self.start_enc_time = timeit.default_timer()
    for i in self.Input_data_in_binry_coded_ascii_format:
        C = (int(i)**e)%n
        self.Encript_Message_ascii_format.append(C)
    Encript_Message_binary_format=[]
    for i in self.Encript_Message_ascii_format:
        temp_=bin(i)[2:]
        Encript_Message_binary_format.append(temp_)
```

3.2.3 Decryption

In the proposed research, decryption is the process of converting encrypted ciphertext back into its original message form. The decryption process is defined by Equation (11):

$$m = c^{d \times kd} mod\ N \tag{11}$$

Here's a detailed explanation of this expression:

1. Ciphertext (c): The ciphertext c is the encrypted data that was obtained from the encryption process. It is the data received by the user, which needs to be decrypted to retrieve the original key.

2. Exponent (*d*): The exponent d is part of the decryption key. Here, d is the private key component that corresponds to the public key component e used during encryption.
3. Decryption Key (*kd*): The term *kd* refers to the decryption key, which might be unique or derived from the unique user key used during the encryption process. This key is crucial for correctly decrypting the ciphertext.
4. Modulus (*N*): The modulus *N* is the same as the one used in the encryption process. It defines the size of the number space for the decryption operation and is typically derived from the product of *N* large prime numbers.

Algorithm for Decryption

- **Step 1:** Read each character from the ciphertext message fileb3. Convert the ASCII value of each character into 8-bit binary and add it in a *fileb4.*
- **Step 2:** Select the block size from the user say it is bs. (Block size should be the same in both encryption and decryption procedures).
- **Step 3:** Calculate no. of bits required to select the binary value from *fileb4*, let's say SZ using the equation *ceil(log2(N)).*
- **Step 4:** Use the decryption key that is generated in section 3.2.1
- **Step 5:** Read the SZ size bits from *fileb4* convert the selected SZ size bits into an equivalent decimal number and store them in a list. A set of decimal numbers will be generated as per the size of SZ.
- **Step 6:** Read decimal values from the list sequentially and decrypt the message using the Equation 11.
- **Step 7:** Then convert the output of step6 to binary and represent each binary bit into *bs* (block size) no of bit and save in a *fileb5*.
- **Step 8:** Read 8-bit data from *fileb5* then convert it into equivalent decimal numbers. Then generate the symbols from the decimal numbers and store them in a *fileb6*.
- **Step 9:** *fileb6* will be our original plain text.

This decryption mechanism ensures that the ciphertext *c* is securely transformed back into the original message mmm. Only authorized users with the appropriate decryption keys *d* and *kd* can perform this operation, thereby maintaining the confidentiality and integrity of the communication. This process is crucial for the overall security of the multi-party authentication scheme, as it guarantees that only legitimate users can access the original message. The Python programming language is utilized to incorporate the

decryption technique. Herein, a snippet of the decryption function's source code is presented:

```
def decrypt_step_4(self):
    last_output_file="file5.txt"
    try:
        dec_file = open(last_output_file,"w")
    except Exception as e:
        print(e+" error occoured when opened "+last_output_file)

    for i in self.the_read_message:
        dec_file.write(i)
    dec_file.close()def decrypt_step_2(self):
    self.Input_data_in_binary_decoded_ascii_format=[]
    for i in self.Encript_Message_ascii_format__temp:
        ttt=i
        m = (ttt**d) % n
        self.Input_data_in_binary_decoded_ascii_format.append(m)
```

The development model illustrated in Figure 1 depicts the key generation and distribution technique among multiple users within a multi-party authentication scheme. The process of generating a unique key for each user is executed during the Key Generation step, designed to accommodate 'K' users. Each user receives a pair of keys: a common key (Key_{pool}) shared

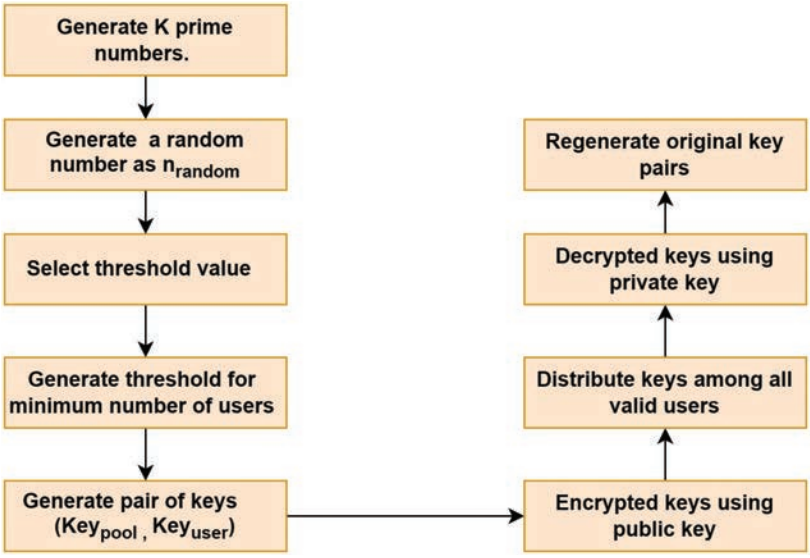

Figure 1 Development model for key generation and distribution among multi-party users.

among all users and a unique key (Key_{user}) specific to each user. The detailed explanation of the proposed key generation and regeneration model is as follows:

1. Selection of Prime Numbers: K prime numbers $(p1, p2, \ldots, pk)$ are selected for generating the key pairs for K users.
2. Random Number Generation: A random number $\mathrm{n_{random}}$ is generated within the specified range.
3. Threshold Selection: A threshold value is selected, representing the minimum number of users required for the authorization of the pool transaction.
4. Unique Key Generation: The unique key for each user Key_{user} (i), is generated using the Equation (5) presented in Section 3.1.
5. Common Key Generation: The common key, Key_{pool}, is generated using the Equation (6) presented in Section 3.1.
6. Key Encryption and Distribution: Encrypt the generated keys as proposed in Section 3.2 and distribute them to the users.
7. Key Decryption by Users: Upon receiving their keys, users decrypt them as described in Section 3.2 and input their keys for authentication.
8. Key Authentication: The authentication process of the keys is performed as described in Section 3.3.

This model appears to be describing a secure key generation and distribution system, possibly for a multi-user cryptographic application and secure key management. It incorporates elements of public-key cryptography, random number generation, and controlled key distribution. It ensures that keys are generated securely, distributed safely to valid users, and can be regenerated as needed.

3.3 Key Authentication and Authorisation

The decrypted keys are used for authentication and authorisation of each user for the initiated transaction to proceed. The decrypted keys that are being received from the user are unique user keys $\mathrm{key_{user}}$ and the common user pool key $\mathrm{key_{pool}}$. From this pair of keys received from the users the authentications is done.

Step 1: Generating $\mathrm{n_{random}}$

The authentication of keys in the proposed multi-party authentication scheme is a multi-step process. In the first stage on receiving the *k* key pairs (key_{useri}, key_{pool}) the n_{random} number is being generated. To generate the n_{random}, the

following Equation (12) is followed

$$n_{random} = GCD(P_i, P_j) \quad where,\ i \neq j, \quad and \quad P_i, P_j \in key_{user} \tag{12}$$

- Greatest Common Divisor (*GCD*): The gcd of two numbers P_i and P_j is the largest positive integer that divides both numbers without leaving a remainder.
- Prime and Integer Combination: When the key pairs are generated, each Key_{user} is a multiplicative combination of a prime p and an integer a∈Z (the set of all integers). Thus, for a particular pool of users, the integer in the multiplicative combination of the Key_{user} is n_{random}.
- When *GCD*(P_i, P_j) is computed, the common factor n_{random} is derived since it is the common multiplicative component in Key_{user}.

Step 2: Computing the Threshold

When the keys pairs were generated for the users the key_{user} is a multiplicative combination of a prime *p* and an integer $a \in Z$, where Z is the Integer Set. Since for a particular pool of users the integer in the multiplicative combination of the key_{user} is the n_{random}, so *GCD*(P_i, P_k) generates the n_{random} where $P_j = p_j \times n_{random}$, where $p_j \in$ *Prime Numbers Set*.

Once the n_{random} is calculated, the minimum number of users required to authenticate the transaction is computed as Equation (13)

$$threshold = key_{pool} \times n_{random}^{-1} \tag{13}$$

- Key_{pool}: The common key shared among all users.
- *Threshold*: This value represents the minimum number of users needed to authorize a transaction, ensuring the integrity and security of the process.

Step 3: Calculating Intermediate Value

Now, that we have the n_{random} and threshold and the number of users u submitted their key pair, the intermediate value is computed using the Equation (14) inspired from Leaky Integrate and Fire model [40].

$$K = U(threshold)/R + Sum(n_{random}) \tag{14}$$

Where R is computed using the formulation in Equation (15)

- Threshold Adjustment: *U*(*threshold*) is a function of the threshold that adjust based on the number of users and the specific threshold required.
- Summation of n_{random}: The sum of all n_{random} values submitted by users, indicating the combined effect of all random numbers in the pool.

Step 4: Calculating the Value of R and u(x)

$$\mathrm{R} = (\mathrm{GCD}(\mathit{key}_{\mathit{useri}}, \mathit{Key}_{\mathit{userj}})/\mathrm{key}_{\mathrm{pool}}) \quad \text{Where i} \neq \text{j} \in \text{ Users} \tag{15}$$

and the function u(x) is computed using the Equation (16)

$$\mathrm{u(x)} = \frac{1}{\mathrm{x}} \times \mathrm{abs}\left(\frac{\mathrm{e}}{2} - \mathrm{a}\right) \quad \text{where a} = \mathrm{e}^{\mathrm{x}/\mathrm{n}_{\mathrm{random}}} \tag{16}$$

- R: is derived by normalizing the GCD of $\mathit{key}_{\mathit{useri}}$ and $\mathit{Key}_{\mathit{userj}}$ with the $\mathit{key}_{\mathit{pool}}$, providing an intermediate value for the authentication process.
- e: Represents a constant or parameter used in the calculation.
- x: A variable used in the calculation, part of the formulation for determining a
- a: Represents a value derived from e and x, crucial for determining $u(x)$.
- u(x): A function that computes a value based on x, typically for validation or verification purposes.

Step 5: Interval for a

In Equation (16), the value of a lines between the open interval of 1 and 2, i.e. $1 < \mathrm{a} < 2$ as can be seen from Equation (17).

$$a = e^{\mathrm{x}/\mathrm{n}_{\mathrm{random}}}$$

$$\log_e a = (\mathrm{x}/\mathrm{n}_{\mathrm{random}})$$

Thereby

$$(\mathrm{x}/\mathrm{n}_{\mathrm{random}}) < 1$$

$$0 \ll \mathit{log}_e a \ll 1$$

$$e^0 \ll a \ll e^1$$

$$1 \ll a \ll 2 \tag{17}$$

For authenticating, where 1 represents a successful authentication and 0 otherwise, a step function is equated using the $\mathrm{n}_{\mathrm{random}}$ and the minimum number of users threshold as per Equation below

$$\mathrm{Authentication}(\mathrm{K}, \mathrm{Key}_{\mathrm{pool}})$$

$$\begin{array}{rll} = 1 & \mathrm{K} \gg \mathrm{Key}_{\mathrm{pool}} & \text{authentication is successful} \\ 0 & \text{else} & \text{authentication is unsuccessful} \end{array}$$

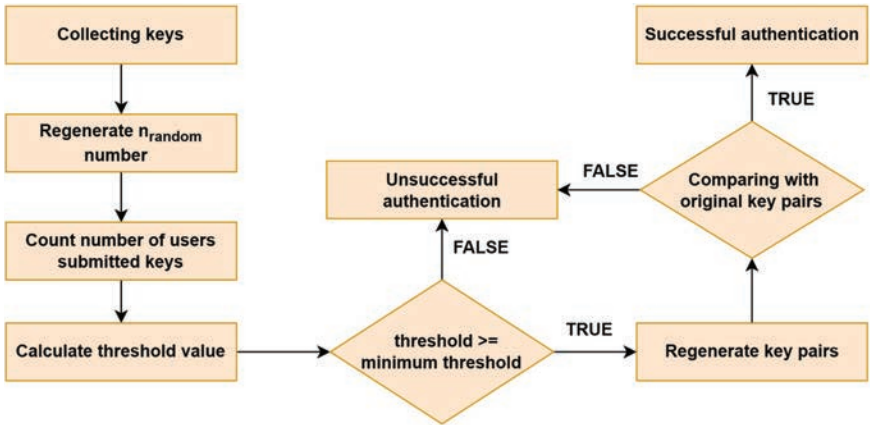

Figure 2 Development model for multi-party authentication technique using distributed key pairs.

- K Number of users who have successfully submitted their valid key pairs.
- Key_{pool}: Minimum number of users required to authenticate the transaction.
- The function returns 1 if K meets or exceeds Key_{pool}, indicating successful authentication. Otherwise, it returns 0, indicating failure.

These equations and the authentication function form a structured approach to ensuring secure authentication in a multi-party environment, leveraging mathematical operations and logical conditions to maintain system integrity and security.

Figure 2 represents the development model for a multi-party authentication scheme using distributed key pairs. The decrypted keys are essential for the authentication and authorization of each user to proceed with the initiated transaction. Each user provides a unique user key (Key_{user}) and a common user pool key (Key_{pool}). The detailed steps for the authentication and authorization process are outlined below:

1. Receive Key Pairs: Collect the K key pairs (Key_{useri}, Key_{pool}) from the users.
2. Regenerate Random Number: Regenerate the random number n_{random} using the Equation (12) presented in Section 3.3.
3. Count User Submissions: Count the number of users who have submitted their keys.

4. Calculate Threshold: Once n_{random} is determined, Calculate the minimum number of users required to authenticate the transaction using Equation (13).
5. Intermediate Value Computation: Compute intermediate values R and $u(x)$ for key verification refer to Equations (15) and (16) in Section 3.3.
6. Threshold Verification: Check if the computed threshold is greater than or equal to the minimum threshold.
   (a) If the condition is false, the authentication is unsuccessful.
   (b) If the condition is true, proceed to regenerate the key pairs.
7. Key Pair Regeneration: Regenerate the key pairs for comparison.
8. Compare Key Pairs: Compare the regenerated key pairs with the original key pairs.
   (a) If they match, return a value of 1, indicating successful authentication.
   (b) If they do not match, return a value of 0, indicating unsuccessful authentication.

This multi-party authentication scheme provides a robust method for verifying user authenticity in distributed systems. The use of both unique user keys and a common pool key adds an additional layer of security. The regeneration process ensures that the system can verify the authenticity of the keys without storing them, reducing the risk of key compromise. This model ensures that the transaction proceeds only if the keys from a sufficient number of users are authenticated successfully, maintaining the integrity and security of the multi-party authentication scheme. Python programming code of the authentication technique are given below.

```
def Authentication(minus,seed,localotps):
    otpsum = 0
    ref = GCD(\
    localotps[random.randint(0,math.floor(len(localotps)/2))],\
    localotps[random.randint(math.floor(len(localotps)/2),
                                        len(localotps)-1)])
    if(len(set(localotps)) == len(localotps)):
        for i in range(0,len(localotps)-1):
            if(GCD(localotps[i],localotps[i+1]) == ref):
                k=GCD(localotps[i],localotps[i+1])
                otpsum = otpsum + GCD(localotps[i],localotps[i+1])
            else:
                return(False)
        if(otpsum >= (seed*(minus-1))):
            return(True)
        else:
            return(False)
    else:
        return(False)
```

4 Result and Discussion

This algorithm has been implemented on several different data files with different types of content and a large range of sizes. In this algorithm, we are test results on text, image, audio, pdf, and exe file with a wide range of sizes and analysis the results of encryption time, decryption time, and key generation time. The specialty of this algorithm is that, if we change the key size and the number of prime numbers then the encryption and decryption time will be different at each time for each type of file.

Table 2 has been formed with the data of the different sizes of the file from 967 KB to 42168 KB. Here it has been clearly visible encryption time is being varied from 0.1144494 to 75.2570445 seconds whereas the decryption times fell in the range 9.1965504 to 364.9413392 seconds.

In Table 3 we show the encryption time and decryption time per byte for a varying range of sizes of the files. Encryption time and decryption time

Table 2 Relation between file size along with the encryption and decryption time

Sl. No	File Name	File Size in (KB)	Encryption Time in (Second)	Decryption Time in (Second)
1	msg1.txt	967	0.1144494	25.8857628
2	image1.png	1,344	0.1827378	9.1965504
3	msg2.txt	2,248	0.1931635	33.3378576
4	image2.png	2,872	1.279183	17.4180415
5	Audio1.ogg	5,629	1.7392502	9.8239028
6	image3.png	5,700	3.1734976	36.8553448
7	msg4.txt	6,359	1.6937247	35.1927572
8	plaintext1.pdf	8,153	2.9109038	12.0046962
9	plaintext2.pdf	8,952	5.483414	63.0005261
10	msg5.txt	10,254	0.4462556	55.8647128
11	Audio2.ogg	10,322	4.7383874	114.2166945
12	Audio3.ogg	12,570	7.3152153	155.6710366
13	image4.png	13,493	6.6243297	6.3726097
14	image5.png	18,231	12.8300466	18.9314566
15	plaintext3.pdf	19,705	17.5951668	33.3748388
16	Audio4.ogg	21,844	27.2614083	32.5343995
17	msgexe1.exe	25,768	36.7238134	265.6873735
18	plaintext4.pdf	26,263	37.2741209	161.9591405
19	plaintext5.pdf	29,557	37.2410171	11.1103561
20	msgexe2.exe	31,400	50.3036763	330.9571857
21	Audio5.ogg	35,363	48.3952452	364.9413392
22	msgexe3.exe	42,168	75.2570445	288.1425202

Table 3 Comparative study between file size, encryption time and chi-square value of encryption

Sl. No	File Name	File Size (KB)	Encryption Time/ Byte for Random Generate Public Key (second)	Encryption Time/ Byte for Particular Value of Public Key (Second)	Chi-square Value
1	msg1.txt	967	0.0001183551	0.0000926560	5742.371295
2	image1.png	1,344	0.0001359656	0.0001376399	13174.74561
3	msg2.txt	2,248	0.0000859268	0.0000989079	36150.73125
4	image2.png	2,872	0.0004453980	0.0001306622	65442.74255
5	Audio1.ogg	5,629	0.0003089803	0.0001428621	15337.78589
6	image3.png	5,700	0.0005567540	0.0001946756	34464
7	msg4.txt	6,359	0.0002663508	0.0000738053	67134.71024
8	plaintext1.pdf	8,153	0.0003570347	0.0003131271	158955.0924
9	plaintext2.pdf	8,952	0.0006125351	0.0003497455	218772
10	msg5.txt	10,254	0.0000435201	0.0000991349	97836
11	Audio2.ogg	10,322	0.0004590571	0.0004336135	107424
12	Audio3.ogg	12,570	0.0005819583	0.0005279775	41115.25356
13	image4.png	13,493	0.0004909457	0.0005397355	312435.0277
14	image5.png	18,231	0.0007037489	0.0007288003	342724.435
15	plaintext3.pdf	19,705	0.0008929290	0.0007863658	424356
16	Audio4.ogg	21,844	0.0012480044	0.0008557902	121992.0938
17	msgexe1.exe	25,768	0.0014251713	0.0009821911	150840
18	plaintext4.pdf	26,263	0.0014192636	0.0010578924	257402.9415
19	plaintext5.pdf	29,557	0.0012599728	0.0012168229	257402.9415
20	msgexe2.exe	31,400	0.0016020279	0.0012679882	305210.7171
21	Audio5.ogg	35,363	0.0013685277	0.0014440643	372021.3979
22	msgexe3.exe	42,168	0.0017846956	0.0018893420	506016

both will be increased according to the file size. In Table 3, the fourth and fifth columns from the left show the time taken for encrypting the particular bytes for each different file. In Table 3, the fourth column from the left shows encrypting time per byte on random generates public key based on ‘e’ valid as range $1 <$ ‘e’ $< \Phi(n)$. Here it has been clearly visible encryption time per byte is being varied from 0.0001183 to 0.0017846 seconds. In the same table, the fifth column from the left shows encrypting time per byte on the particular value of ‘e’ in the public key as per the algorithm. Here it has been clearly visible encryption time per byte is being varied from 0.0000926561 to 0.0018893420 seconds. In this table, we also mention the chi-square value in column no. sixth from the left side. The value of chi-square significantly varies from 5742.371295 to 506016.

Table 4 Key generation and regeneration time based on number of prime numbers

Number of Prime Numbers	Key Generation Time (Second)	Key Regeneration Time (Second)
2	0.0643183000	0.0000318000
3	0.0772252000	0.0000392000
4	0.1576525000	0.0001568000
5	0.0745648000	0.0000483000
6	0.0754701000	0.0000519000
7	0.1910504000	0.0001304000
8	0.0805093000	0.0000594000
9	0.0738529000	0.0000648000
10	0.0823818000	0.0000968000

Table 5 Key generation and regeneration time based on number of digits of prime number

Sl.no	Number of Digits of Prime Numbers	Key Generation Time (Second)	Key Regeneration Time (Second)
1	2	0.0868604000	0.0000660000
2	3	0.1063464000	0.0000626000
3	4	0.0894444000	0.0000737000
4	5	0.0802372000	0.0000854000
5	6	0.0886877000	0.0001074000
6	7	0.0954053000	0.0001087000
7	8	0.1040706000	0.0001299000
8	9	0.1227227000	0.0001333000
9	10	0.2263527000	0.0001400000

In Table 4 the second column from the left shows the time it takes to generate the keys at the time of encryption. Whereas regenerating the specific key at the time of decryption from the set of distributed keys is required. The number of prime numbers in the range of 2 to 10 is shown in the first column from the left of this table. The key generation time based on the value of the first column from the left in the same table has ranged from 0.0643183000 to 0.1910504000 seconds. Table 4 shows the corresponding regeneration time of the key based on the value of the first column in the third column from the left with a fluctuation of 0.0000318000 to 0.0001568000 seconds.

Table 5 shows important key generation and regeneration times based on the number of digits in prime numbers. The second column from the left in this table shows the number of digits of prime numbers in the range of two to ten. In the same table, the key generation time is listed in the third column from the left. It has fluctuated between 0.0802372000 and 0.2263527000

seconds. The key regeneration time is indicated in the fourth column from the left in Table 5. It has been varying from 0.0000626000 to 0.0001400000 seconds.

5 Comparison and Performance Analysis

In this section, we have analyzed all of the results of section 4 of all different file sizes, file types, encryption times, and key generation times. We also use a frequency distribution graph and chi-square value to analysis all results.

A graphical representation for Table 3 is shown in Figure 3 with a continuous solid line and dotted line. In this figure dotted line indicates encrypting timeper byte on random generates a public key based on 'e' validated as range $1 <$ 'e' $< \phi$(n) and the solid line indicates the encrypting time per byte on the particular value of 'e' in the public key as per the algorithm.

It has been observed that the two lines in the graph are almost the same in nature. It is actually reflected much more difference in times per byte does not happen for the randomly selected value of 'e' and the specific value of 'e'. The graph also shows times per byte are increased exponentially with the incremented size of the file. So after a certain file size implementation of this algorithm takes a huge amount of time during the encryption. This algorithm increases encryption time per byte exponentially in both cases for the randomly selected value of 'e' and the specific value of 'e'. However, generating 'e' at random improves the algorithm's security. The analysis of

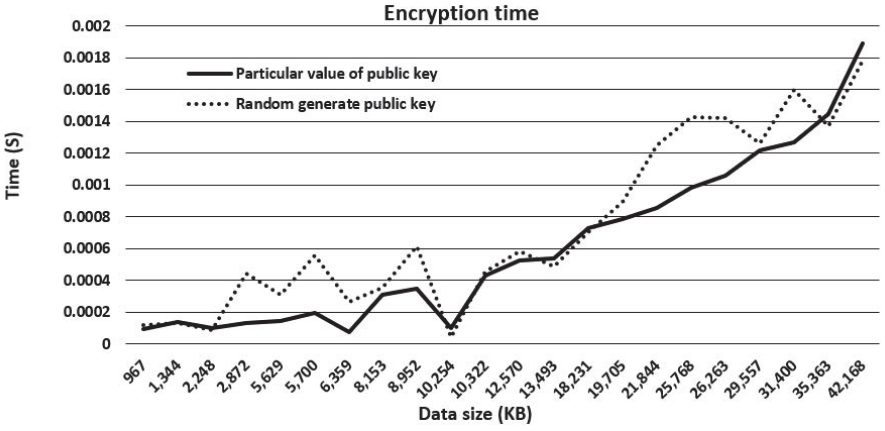

Figure 3 File size versus encryption time per byte.

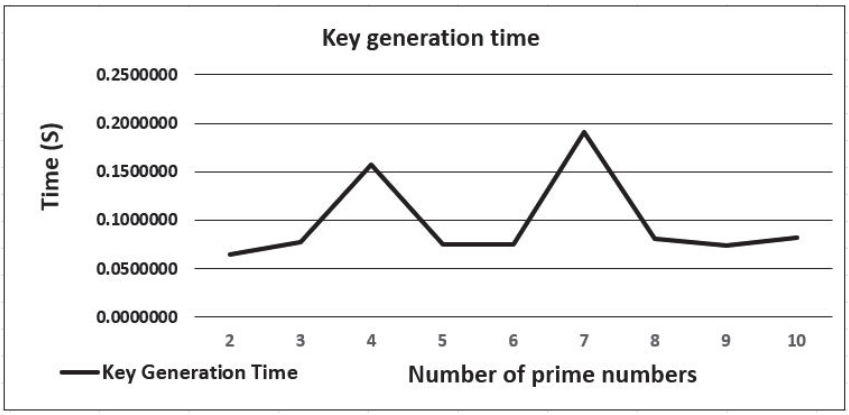

Figure 4 Key generation time based on number of prime numbers.

time complexity is done to show the feasibility of the implementation of our proposed method.

A graphical representation for Table 4 is shown in Figure 4 with a continuous solid line. In this figure, the solid line indicates the key generation time on a number of prime numbers. According to the graph, there is a tendency for key generation time to fluctuate between 0.05000000 to 0.2000000 seconds. Here, it generates prime numbers randomly that are also affected in the graph. It is analyzed that if the number of prime numbers increases then key generation time sometimes increases and also sometimes decreases because key generation time depends on the value of prime numbers but not the number of the prime numbers.

Figure 5 shows a continuous solid line as a graphical representation of Table 5. The solid line in this diagram represents the key generation time for a number of digits of prime numbers. Here it is considered six prime numbers. According to the graph, the time it takes to generate a key gradually increases. It generates prime numbers randomly, which have an impact on the graph. In this graph, it shows key generation time gradually increases after a certain number of digits of the prime number. It has been observed from the graph that key generation time exponentially increases after more than eight digits of the prime numbers. There are some problem may arise regarding time management if we used prime numbers having more than ten digits in our algorithms.

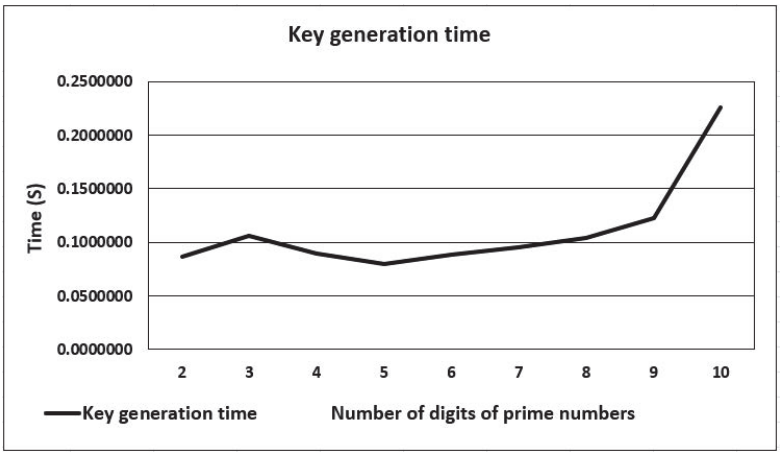

Figure 5 Key generation time based on number of digits of prime number.

Chi-Square Test

Here we use the "Pearsonian Chi-Square Value," also known as the "Goodness-of-Fit Chi-Square Test," to check the non-homogeneity of the source file and the matching encrypted file, using the formula $\mathrm{x}^2 = \sum (f0 - fn)^2$/fn. The frequency of a character in the source file and the frequency of the same character in the associated encrypted file are represented by f0 and fn, respectively. The Chi-square values for sample pairs of all sizes and types of files have been determined using this chi-Square formula, and the chi-square values of all encrypted files are listed in the sixth column from the left of Table 3. We get very high Chi-square values for all type of files. This high chi-square values indicate that there is very little similarity have between the original and encrypted data. This procedure is safer as the value of Chi-Square high are very high, and it is a widely accepted technique in the field of cryptography.

Graphical Test for Frequency Distribution

The objective of this frequency distribution is to determine whether there exists a consistent relationship between characters in both the source and encrypted files. This evaluation aims to measure the security level of the proposed method against cryptanalytic attacks. It involves a graphical comparison of the frequency distribution of all 256 characters in the source file and their equivalent characters in the encrypted file.

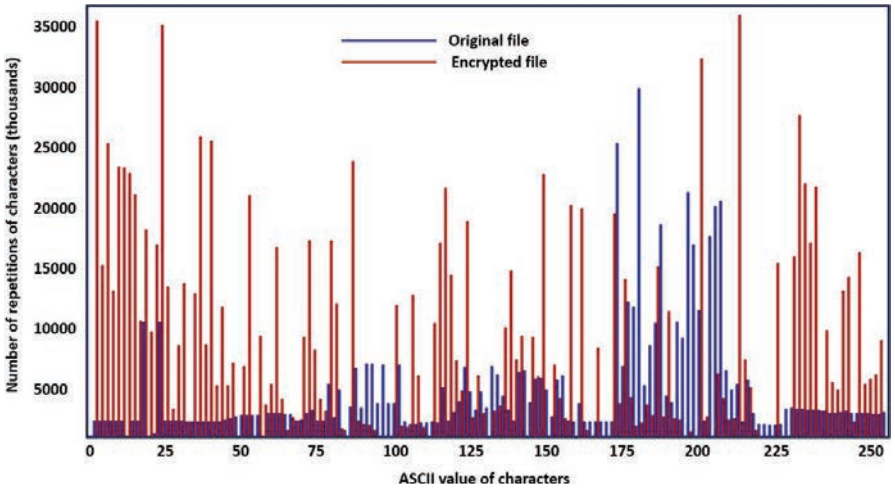

Figure 6 Key generation time based on number of digits of prime number.

Comparison

This section conducts a comprehensive comparison and analysis of encryption and decryption times across original RSA, modified RSA, and proposed RSA methodology. Tables 6 and 7 outline the performance metrics for the original RSA algorithm developed by Rivest, Shamir, and Adleman [3], a modified version of RSA [22], and the newly proposed N prime RSA algorithm. Table 6 specifically focuses on encryption time results, while Table 7 presents findings related to decryption time. Furthermore, a visual representation of the performance, encapsulating the encryption and decryption times for the original RSA, Modified RSA, and N prime RSA schemes is illustrated in Figures 7 and 8, respectively.

Table 6 Comparison of encryption between RSA, MRSA and N Prime RSA

File Size in (KB)	Traditional RSA (Second)	Modified RSA (Second)	N Prime Modified RSA (Second)
278	0.0048	0.007	0.023612848
569	0.0082	0.0091	0.037225697
923	0.0102	0.0191	0.080838545
1130	0.0175	0.0344	0.114451394
1435	0.0283	0.0626	0.128064242
1728	0.0339	0.0792	0.146770904
1977	0.0461	0.0905	0.165289939

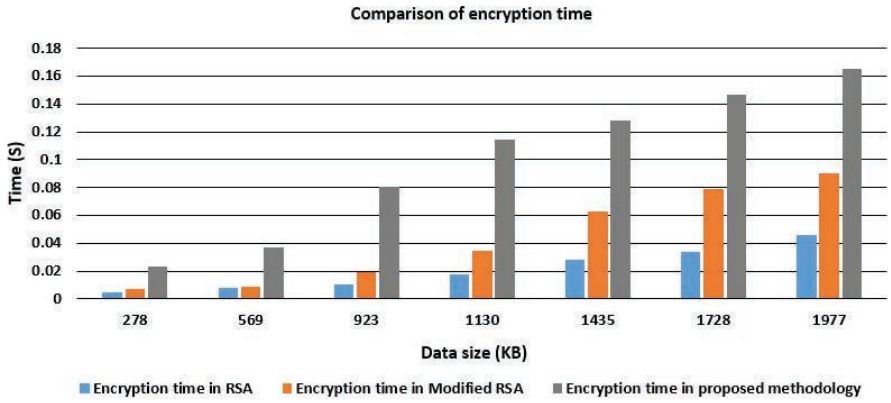

Figure 7 Encryption time comparison between RSA, MRSA and N Prime RSA.

Table 7 Comparison of Decryption between RSA, MRSA and N Prime RSA algorithm

File Size in KB	Original RSA (Second)	Modified RSA (Second)	N Prime Modified RSA (Second)
278	0.0319	0.0818	0.65898
569	0.0632	0.1569	1.02389
923	0.079	0.2344	1.25498
1130	0.0946	0.3422	1.52671
1435	0.1098	0.4027	1.71652
1728	0.1267	0.4863	1.85815
1977	0.1547	0.6716	1.96769

Table 6 and Figure 7 display a comparative analysis of encryption times. The graphical representation indicates a noticeable increase in encryption time with a corresponding rise in file size. It is evident from the analysis that the proposed methodology exhibits a higher encryption time compared to both the original RSA and the modified RSA. This extended encryption time in the proposed N prime RSA technique can be viewed as advantageous, as the added complexity contributes to heightened system resilience, making it more challenging to breach within a reasonable timeframe.

Table 7 provides a comprehensive overview of the decryption times for RSA, Modified RSA, and the proposed methodology, with a corresponding visual representation in Figure 8 using a bar chart. The graphical comparison highlights the escalating decryption times as file sizes increase, showcasing the efficiency disparities among RSA, Modified RSA (MRSA), and the proposed N prime RSA techniques.

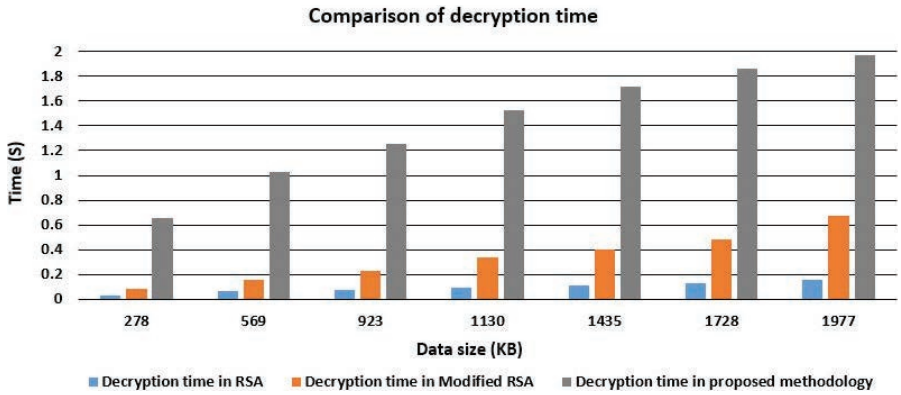

Figure 8 Decryption time comparison between RSA, MRSA and N Prime RSA.

The observed results indicate a notable extension in decryption time for the proposed methodology compared to both RSA and Modified RSA algorithms. For example, with an input file size of 1130 KB, the original RSA demonstrates a decryption time of 0.0946 seconds, the Modified RSA takes 0.3422 seconds, while the proposed N prime RSA methodology requires 1.52671 seconds.

The graphical representations in Figures 7 and 8 collectively underscore that both encryption and decryption times surpass those of RSA and Modified RSA. This increase in time is deemed acceptable due to its proportional enhancement in security within the proposed N prime RSA methodology.

6 Security Analysis and Application

The proposed N prime RSA methodology is widely used in the hospitality industry and enables secure communication over an insecure network. Securing a Hotel Management System (HMS) is crucial to protect sensitive guest information, booking information, and financial transactions, and ensure the smooth operation of the business. The proposed N RSA methodology ensures higher security of hospitality data. Security in the proposed techniques relies on the difficulty of factoring the product of randomly generated N large prime numbers. It is also directly related to the size of the key_{pool}, key_{user}, and threshold value which generate some higher mathematical computation suck as a prime number, greatest common divisor, Euler talent function, modular arithmetic, etc. The proposed methodology generates random prime

numbers to prevent attackers from predicting or influencing the prime selection. The suggested encryption techniques encryption typically use padding schemes to add randomness and structure to the plaintext before encryption. Padding helps prevent certain types of attacks, such as those based on the mathematical properties of the proposed methodology. The implementation of the suggested must be resistant to various side-channel attacks, such as timing attacks and power analysis attacks as the proposed methodology includes multi-party authentication techniques. The security of the suggested technique is increased as compared to the original RSA and modified RSA as suggested techniques take larger encryption and decryption time.

Applications in the Hospitality Industry

A Hotel Management System (HMS) is a comprehensive web application solution designed to streamline and automate the operations of the hospitality industry. It encompasses various modules and functionalities to help manage different aspects of hotel operations efficiently. Securing a hotel management system is crucial to protect sensitive guest information, and financial transactions, and ensure the smooth operation of the business. The proposed N prime RSA cryptography technique plays a crucial role in securing various aspects of a hotel management system, especially when it comes to protecting sensitive data such as guest information, reservations, and financial transactions.

Table 8 Comparison of existing hospitality industry applications

Author	Methodology	Finding Application Area
A.K. Tripathy [39]	RSA Methodology	Smart tourism
S. Cheong et al. [40]	Encrypted Steganography Quick Response (ESQR) key	Hotel access control system
Y. Ge et al. [41]	Elliptical curve cryptography and block cipher algorithm	Data security protection system
Manish Verma [42]	Blockchain Technology	Hostel room booking system
Aditya Kumar Sahu et al. [43]	grayscale steganography	Hotel services
M. Dammak et al. [44]	Internet of Things.	Smart hotels
J. T. Wosu et al. [45]	Embedded IoT-Enabled Database Management System	Hotel room reservation accountability

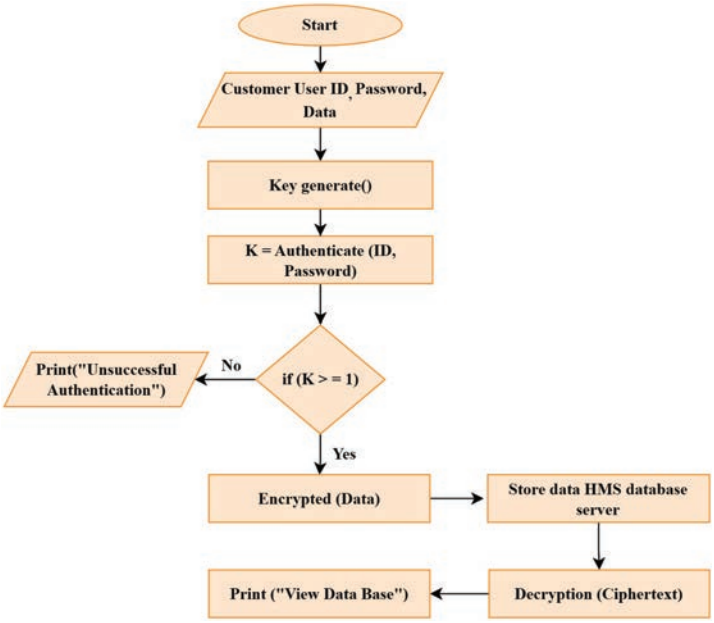

Figure 9 Workflow diagram of N prime modified RSA in Hospitality Industry.

HMS used the proposed multi-party authentication and authorization technique for secure user authentication. It provides higher security in the authentication process. Only authorized users can authenticate in the system. The proposed encryption methodology is used to encrypt customer data such as guest information, reservation data, credit card data, and financial data. As data is transferred unreadable format it is very difficult for an attacker to read the content of the data. The suggested decryption process is used to convert data unreadable to readable using our proposed N Prime RSA algorithm. These high secure key generation, encryption, and decryption techniques provide higher security in the hospitality industry to enhance the security of the customer data.

7 Conclusion

This work emphasizes the crucial importance of authentication and authorization techniques in ensuring secure transactions over the Internet, especially in high-risk defense systems, digital financial systems, and hospitality management systems. While single-user authorization methods have been prevalent,

this work emphasizes the need for more robust multi-user authorization techniques. In this paper, it has been presented a novel multiparty key distribution method. Our experimental results show that the proposed work successfully generates and validates keys for multiple users. In addition, it can easily be coupled with any cryptography technique. This innovative methodology not only bolsters security but also improves the efficiency of multi-party authentication processes. The results of this work demonstrate notable improvements in time complexity and resilience against various forms of attacks, underlining the promising prospects of these methods in enhancing online security. The proposed multi-user authentication, encryption, and decryption techniques are used in the hospitality industry to enhance higher security of data transformation.

However, it is important to acknowledge a limitation in the proposed method is encryption time increases exponentially. It is observed that the encryption time exhibits exponential growth as the file size expands which is leading to significant delays in the encryption process for larger files. To address this challenge, a promising avenue for future research entails devising strategies to reduce encryption time, particularly for substantial file sizes.

Declarations

Ethical Approval

Not applicable

Competing Interests

Author Bilas Haldar declares that he has no conflict of interest. Author Rohit Sinha declares that he has no conflict of interest. Author Dr. Pranam Paul declares that he has no conflict of interest.

References

[1] A. A. Hajomer, I. Derkach, N. Jain, H.-M. Chin, U. L. Andersen, T. Gehring, Long-distance continuous-variable quantum key distribution over 100-km fiber with local local oscillator, Science Advances 10 (1) (2024) eadi9474.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (7779) (2019) 505–510.

[3] S. B. Das, S. K. Mishra, A. K. Sahu, A new modified version of standard rsa cryptography algorithm, in: Smart Computing Paradigms: New Progresses and Challenges, Springer, 2020, pp. 281–287.

[4] F. O. Mojisola, S. Misra, C. F. Febisola, O. Abayomi-Alli, G. Sengul, An improved random bit-stuffing technique with a modified rsa algorithm for resisting attacks in information security (rbmrsa), Egyptian Informatics Journal (2022).

[5] N. M. Al-Jubouri, R. J. S. Al-Janabi, Secure rsa cryptosystem based on multiple keys, Journal of Al-Qadisiyah for computer science and mathematics 13 (3) (2021) Page–25.

[6] G. Kbar, W. Mansoor, Modified rsa using triple keys based encryption/decryption, Jordan Journal of Electrical Engineering. All rights reserved-Volume 7 (1) (2021).

[7] B. B. Ahamed, M. Krishnamoorthy, Sms encryption and decryption using modified vigenere cipher algorithm, Journal of the Operations Research Society of China (2020) 1–14.

[8] M. Wasim, P. Paul, Implementing the information security using modified rsa algorithm with the help of n prime number, International Journal of Innovative Research in Computer and Communication Engineering 4 (10) (2016) 18055–18062.

[9] R. Ghosh, An efficient and robust modified rsa based security algorithm in modern cryptography, Journal of Computer Science Engineering and Information Technology Research (JCSEITR) 6 (2) (2016) 15–22.

[10] N. Kumar, P. Chaudhary, Implementation of modified rsa cryptosystem for data encryption and decryption based on n prime number and bit stuffing, in: Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, 2016, pp. 1–6

[11] W.-K. Yu, N. Wei, Y.-X. Li, Y. Yang, S.-F. Wang, Multi-party interactive cryptographic key distribution protocol over a public network based on computational ghost imaging, Optics and Lasers in Engineering 155 (2022) 107067.

[12] C. Wang, H. Zhu, A secure centralized multi-party quantum key distribution protocol with new encoding mode, International Journal of Theoretical Physics 62 (6) (2023) 128.

[13] S. Kumar, D. Kumar, Securing of cloud storage data using hybrid aes-ecc cryptographic approach, Journal of Mobile Multimedia (2023) 363–388.
[14] Z. Ashraf, A. Sohail, M. Yousaf, Robust and lightweight symmetric key exchange algorithm for next-generation ioe, Internet of Things 22 (2023) 100703.
[15] V. Lovic, Quantum key distribution: Advantages, challenges and policy (2020).
[16] S. Kleis, C. G. Schaeffer, Improving the secret key rate of coherent quantum key distribution with bayesian inference, Journal of Lightwave Technology 37 (3) (2018) 722–728.
[17] Y. Kang, S. Kanwal, B. Liu, D. Zhang, Ghost key distribution under mutual authentication mechanism, Information Sciences 640 (2023) 119025.
[18] Y. Ramadhan, S. Suhardi, Y. Aditama, Data security using low bit encoding algorithm and rsa algorithm, Jurnal Mantik 8 (1) (2024) 16–25.
[19] S. P. Jadhav, Towards light weight cryptography schemes for resource constraint devices in iot, Journal of Mobile Multimedia (2019) 91–110.
[20] J.-W. Zhang, G. Xu, X.-B. Chen, Y. Chang, Z.-C. Dong, Improved multiparty quantum private comparison based on quantum homomorphic encryption, Physica A: Statistical Mechanics and its Applications 610 (2023) 128397.
[21] S. Bhasin, J.-P. D'Anvers, D. Heinz, T. Pöppelmann, M. Van Beirendonck, Attacking and defending masked polynomial comparison for lattice-based cryptography, IACR Transactions on Cryptographic Hardware and Embedded Systems (2021) 334–359.
[22] W.-K. Yu, Y. Yang, Y.-X. Li, N. Wei, S.-F. Wang, Multi-party cryptographic key distribution protocol over a public network based on a quick-response code, Sensors 22 (11) (2022) 3994.
[23] F. O. Mojisola, S. Misra, C. F. Febisola, O. Abayomi-Alli, G. Sengul, An improved random bit-stuffing technique with a modified rsa algorithm for resisting attacks in information security (rbmrsa), Egyptian Informatics Journal 23 (2) (2022) 291–301.
[24] J. K. Dawson, F. Twum, J. B. Hayfron-Acquah, Y. M. Missah, B. B. K. Ayawli, An enhanced rsa algorithm using gaussian interpolation formula, International Journal of Computer Aided Engineering and Technology 16 (4) (2022) 534–552.

[25] F. Zhao, J. Guo, L. Zhang, W. Han, Research on improved double rsa algorithm based on rsa, in: International Conference on Computer Engineering and Networks, Springer, 2022, pp. 1204–1211.

[26] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, R. Cammarota, Post-quantum lattice-based cryptography implementations: A survey, ACM Computing Surveys (CSUR) 51 (6) (2019) 1–41.

[27] M. Gayathri, C. Malathy, A deep learning framework for intrusion detection and multimodal biometric image authentication, Journal of Mobile Multimedia (2022) 393–420.

[28] R. Omollo, A. Okoth, Factorization algorithm for semi-primes and the cryptanalysis of rivest-shamir-adleman (rsa) cryptography, Asian Journal of Research in Computer Science 17 (6) (2024) 85–95.

[29] J. Jin, K. Kim, 3d cube algorithm for the key generation method: applying deep neural network learning-based, IEEE Access 8 (2020) 33689–33702.

[30] J. Shi, S. Chen, Y. Lu, Y. Feng, R. Shi, Y. Yang, J. Li, An approach to cryptography based on continuous-variable quantum neural network, Scientific reports 10 (1) (2020) 1–13.

[31] P. E. Frenkel, J. Pelik´an, On the greatest common divisor of the value of two polynomials, The American Mathematical Monthly 124 (5) (2017) 446–450.

[32] S. Nasreen, A. H. Mir, Enhancing cloud forensic investigation system in distributed cloud computing using dk-cp-ecc algorithm and ek-anfis, Journal of Mobile Multimedia (2023) 679–706.

[33] S. Deshpande, S. M. del Pozo, V. Mateu, M. Manzano, N. Aaraj, J. Szefer, Modular inverse for integers using fast constant time gcd algorithm and its applications, in: 2021 31st International Conference.

[34] D. M. Ghadi, Improving the robustness of rsa encryption through input-based key generation., Mathematical Modelling of Engineering Problems 11 (1) (2024).

[35] J. Liu, Y. Yu, B. Yang, J. Jia, Q. Lai, Cryptanalysis of cramershoup like cryptosystems based on index exchangeable family, International Journal of Foundations of Computer Science 32 (01) (2021) 73–91.

[36] S. Basu, M. Kule, H. Rahaman, Implementation of symmetric functions using memristive nanocrossbar arrays and their application in cryptography, Journal of Circuits, Systems and Computers 30 (12) (2021) 2150223.

[37] M. Rashid, M. Imran, A. R. Jafri, T. F. Al-Somani, Flexible architectures for cryptographic algorithms—a systematic literature review, Journal of Circuits, Systems and Computers 28 (03) (2019) 1930003.

[38] X. Shang, W. Shan, X. Liu, Design and implementation of a reconfigurable cryptographic coprocessor with multiple side-channel attacks countermeasures, Journal of Circuits, Systems and Computers 27 (11) (2018) 1850180.

[39] A. K. Tripathy, P. K. Tripathy, N. K. Ray, S. P. Mohanty, itour: The future of smart tourism: An iot framework for the independent mobility of tourists in smart cities, IEEE consumer electronics magazine 7 (3) (2018) 32–37.

[40] S.-N. Cheong, H.-C. Ling, P.-L. Teh, Encrypted steganography quick response scheme for unified hotel access control system, in: 2021 IEEE 11th International Conference on System Engineering and Technology (ICSET), IEEE, 2021, pp. 303–308.

[41] Y. Ge, W. Xu, L. Zhang, Integrative security protection system for full life cycle of big data based on sm crypto algorithm, in: International-Conference on Algorithms, High Performance Computing, and Artificial Intelligence (AHPCAI 2021), Vol. 12156, SPIE, 2021, pp. 396–402.

[42] M. Verma, Implementation of blockchain-based technique to a hostel room booking system: practical aspects, International Journal for Research in Applied Science and Engineering Technology 9 (5) (2021) 1–4.

[43] A. K. Sahu, A. Gutub, Improving grayscale steganography to protect personal information disclosure within hotel services, Multimedia Tools and Applications 81 (21) (2022) 30663–30683.

[44] M. Dammak, S. Aroua, S. M. Senouci, Y. Ghamri-Doudane, G. Suciu, M.-A. Sachian, R. Roscaneanu, M. O. Gungor, et al., A secure and interoperable platform for privacy protection in the smart hotel context, in: 2020 Global Information Infrastructure and Networking Symposium (GIIS), IEEE, 2020, pp. 1–6.

[45] J. T. Wosu, G. C. Ononiwu, T. C. Oguichen, O. Opara, Development of embedded iot-enabled database management system for improved hotel room reservation accountability. "International Journal of Scientific Engineering and Science", 3 (5), 2019, pp. 36–42.

Biographies

Bilas Haldar working as an Assistant Professor in the Department of Computer Science and Engineering at The Neotia University. He is pursuing a Ph.D. in Computer Science & Engineering at The Neotia University. Mr. Haldar obtained his M.Tech in Software Engineering from Maulana Abul Kalam Azad University of Technology. He has more than 11 years of academic, teaching, and research experience. His research interests include Cyber Security, Cryptography, Network Security, Data Security, Application Security and Machine learning.

Rohit Sinha is a dynamic AI Engineer with a strong foundation in Cyber Security, holding a B.Tech degree in the field. Currently, he is pursuing his M.Tech in Artificial Intelligence from the prestigious BITS Pilani, further honing his skills in the realm of AI. Rohit's areas of interest include Cryptography, Computer Vision, Graph Learning, and Reinforcement Learning, where he aims to leverage his expertise to drive innovation and solve complex problems. With a unique blend of security and AI knowledge, Rohit is poised to make significant contributions to the rapidly evolving tech landscape.

Pranam Paul having the experience of more than 19 years in teaching in different college and university started his career from 2005. He did his Ph.D. in the field of engineering from National Institute of Technology, (Durgapur), West Bengal, India, an institute of national importance in India. He has more than 100 research publication. He is continuing his research in the field of Data and Application Security, Database Management System, Computer Graphics, Machine Learning and also some various field.