Volume 9, Issue 5, May – 2024 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY675

HJ Semi Symmetric Cryptographic Algorithm

Network Security and Cryptography

James HAKIZIMANA[1], Master's
Supervisors: Dr. Wilson MUSONI, PhD and Dr. Emmanuel BUGINGO, PhD
Master of Science in Information Technology (MSCIT), University of Kigali, Rwanda

Abstract:- This paper presents an algorithm for performing encryption and decryption using modular exponentiation mainly based on Euler's Theorem. The algorithm uses the properties of modular arithmetic to ensure secure communication. The gcd function is employed to find coprime numbers within a given range, which are then utilized as bases for encryption and decryption. The modular Exponentiation function efficiently computes the exponentiation of a base to a power modulo a given modulus. Through a step-by-step process, the algorithm encrypts a numerical message using the selected base and then decrypts it, ensuring the original message is recovered accurately. Experimental results demonstrate the effectiveness and reliability of the algorithm for secure data transmission. In the conclusion we discussed the effectiveness of the algorithm compared to the existing ones such as Elgamal algorithm and RSA Cryptography. The algorithm presented in this paper publishes only the modulus (integer) number and protocols of key exchanges kept secret between sender and receiver; therefore, the algorithm is not fully asymmetric but semi symmetric.

Keywords:- *RSA, ECC, GCD.*

I. INTRODUCTION

Protected communication is essential in various domains such as cybersecurity, finance, and data transmission. One fundamental aspect of secure communication is encryption, where sensitive information is encoded to prevent unauthorized access. Cryptographic algorithms are a widely-used method for achieving secure communication, where in encryption and decryption keys are different yet mathematically related.

The algorithm presented in this paper builds upon the principles of public-key cryptography, specifically focusing on modular arithmetic and the gcd (greatest common divisor) function.

The GCD function plays a crucial role in selecting appropriate a base for encryption and decryption. By finding coprime numbers within a specified range, the algorithm ensures that the keys used for encryption and decryption are mutually exclusive yet compatible, enhancing the security of the communication process.

Through the integration of modular exponentiation and careful selection of coprime bases, the algorithm provides a powerful framework for encrypting and decrypting numerical messages. The step-by-step approach outlined in the algorithm ensures both security and efficiency in the encryption and decryption process, making it suitable for a wide range of applications in communication.

- *Statement of Research Question*

How can secure communication be facilitated through the implementation Euler's theorem in the context of public-key cryptography not only for prime number but also for a composite integer?

II. LITERATURE REVIEW

The algorithm presented in this paper draws upon several key concepts and techniques from existing literature in the fields of cryptography and computational number theory. Modular exponentiation and the gcd function are fundamental components of some cryptographic algorithms.

Modular exponentiation is a well-established technique used in numerous cryptographic systems, including RSA (Rivest-Shamir-Adleman) encryption [1]. It involves efficiently computing large powers of a base modulo a given modulus, thereby enabling secure encryption and decryption processes. The modular exponentiation is used to generate encrypted and decrypted messages, ensuring confidentiality and integrity in communication.

The gcd function, has been extensively studied in number theory and computational mathematics. Its primary role in cryptography lies in key generation, where it helps identify coprime numbers essential for constructing secure encryption and decryption keys. By finding numbers with no common factors other than 1, the gcd function facilitates the creation of a most secured cryptographic systems resistant to attacks.

An efficient way to compute $\gcd(a, b)$ is the Euclidean algorithm [2], which many of us have probably already seen. The idea is to use division with remainder. Thus, first we divide a by b and get a quotient q and remainder r.

In the other words,

$a = bq + r$, with $0 \leq r_2 < b$
$b = r_2 q_2 + r_3$, with $0 \leq r_3 < r_2$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY675

$r_3 = r_3q_3 + r_4$, with $0 \leq r_4 < r_3$

$r_{n-1} = r_nq_n + r_{n+1}$, with $0 \leq r_{n+1} < r_n$

$r_n = r_{n+1}q_{n+1}$

Let $r_o = a$ and $r_1 = b$ then we have $b = r_1 > r_2 > r_3 > \cdots$and the r_i's are non negative integers, finally we reach to zero and let say $r_{n+2} = 0$. Then the result is $\gcd(a, b) = r_{n+1}$

➢ *Definition:*

Euler's phi (or totient) function of a positive integer *n* is the number of integers in {1,2,3, ..., *n*} which are relatively prime or coprime to *n*. This is usually denoted φ(*n*).

For prime number *p*, **φ(*p*)=*p*-1**. [3]

➢ *Euler's Theorem:*

If n and a are coprime positive integers, then $a^{\varphi(n)}$ is congruent to 1 modulo n where $\varphi(n)$ denotes Euler's totient function. In mathematical notation we can denote the theorem as follow:

$a^{\varphi(n)} \equiv 1\ mod(n)$ [4]

➢ *Fermat's Little Theorem:*

If p is a prime number and a is an integer not divisible by p, then a^{p-1} is congruent to 1 modulo p, or in mathematical notation:

$a^{p-1} \equiv 1\ (mod\ p)$ [3].

III. METHODOLOGY

Since the provided algorithm appears to be a practical implementation rather than a research study, it doesn't follow a traditional research methodology with hypotheses, data collection, and analysis. However, if we were to frame the methodology for researching or evaluating the algorithm, it might involve the following steps:

➢ *Literature Review:*

Conduct a review of existing literature on symmetric and asymmetric cryptography, modular arithmetic, and related algorithms.

➢ *Algorithm Analysis:*

Analyse the algorithm's complexity, correctness, and security properties. Assess its performance in terms of computational efficiency and scalability, considering factors such as time complexity and memory usage.

➢ *Implementation:*

Implement the algorithm in a programming language, ensuring adherence to the algorithm's specifications and correct handling of inputs and outputs.

➢ *Benchmarking:*

Compare the algorithm's performance against other cryptographic algorithms or implementations. Measure factors such as encryption and decryption speed, memory usage, and resistance to known attacks.

➢ *Evaluation:*

Evaluate the algorithm's suitability for practical applications based on the testing and benchmarking results. Consider factors such as ease of use, compatibility with existing systems, and overall security guarantees.

➢ *Documentation and Reporting:*

Document the research process, including the algorithm's design, implementation details, test results, and conclusions. Prepare a comprehensive report or paper summarizing the research findings and recommendations.

IV. RESULTS

A. Algorithm Structure

- Choose a positive integer n and shift integers $k_1$and k_2 such that $0 < k_1, k_2 < n$
- Choose base a such that $\gcd(a, n) = 1$
- Compute $a_1 = a + k_1$
- Choose an integer $e \in [1,\ \varphi(n)]$
- Compute $e_1 = e + k_2$
- Encryption function $\mu(x) = x \times a^e\ modulo\ n$
- Recipient computes $b = a_1 - k_1,\ (a = b)$
- Recipient computes $d = \varphi(n) - (e_1 - k_2)$
- Decryption function $\lambda(y) = y \times b^d\ modulo\ n$

Symmetric (private) keys will be k_1 and k_2. Public keys will be n, a_1 and e_1.

Notice: Shift keys integers $k_1$and k_2 can be used for different integer n as long as

$0 < k_1, k_2 < n.$

B. Keys Distribution

➢ *Initial Keys*

- Sender and recipient secretly agreed on common integer shift k_1 for position of the base a.
- Sender and recipient secretly agreed on common integer shift k_2 for exponent.

➢ *Encryption Key*

- Choose base a such that $\gcd(a, n) = 1$
- Compute $a_1 = a + k_1$
- Choose an integer $e \in [1,\ \varphi(n)]$.
- Compute $e_1 = e + k_2$.

Encryption key will be $a^e\ mod\ n$ which is kept private.

Public keys (to be sent to the recipient) will be n, a_1 and e_1.

➢ *Decryption Key*

International Journal of Innovative Science and Research Technology
https://doi.org/10.38124/ijisrt/IJISRT24MAY675

- Recipient computes $b = a_1 - k_1$. $(a = b)$
- Recipient computes $d = \varphi(n) - (e_1 - k_2)$.
- Decryption function $\lambda(y) = y \times b^d \ modulo \ n$

Decryption key will be $b^d \ modulo \ n$.

Notice: $e + d = \varphi(n)$ in such way $a^e \times b^d = a^e \times a^d = a^{e+d} = a^{\varphi(n)} \equiv 1 \ modulo \ n$

C. Flowchart of the Algorithm

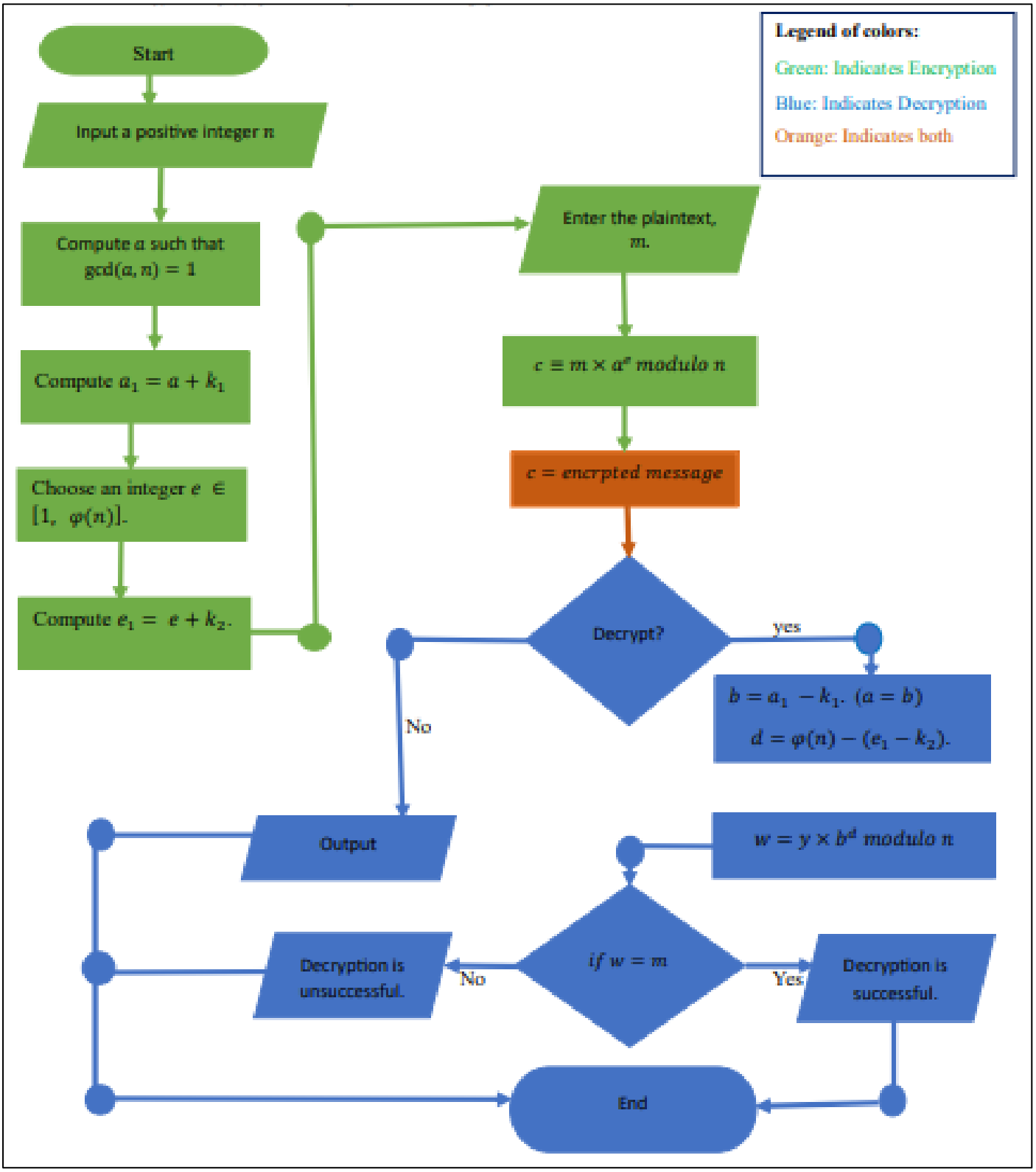

Fig 1 Flowchart of the Algorithm

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY675

D. Pseudocodes

- $Start$
- $Print$ "*Enter a Modulus number*"
- $Read\ N$
- $Print$"$Enter\ a\ base\ shift$"
- $Read\ k_1$
- $Print$"$Enter\ an\ exponent\ shift$"
- $Read\ k_2$
- $for\ j = 3; j \leq N; j++$
- $if\ \gcd(N, j) = 1$
- $print$ "possible base:", j
- $print$ " Choose an appropriate integer from the list provided above to be your base :"
- $Read\ a$
- $a_1 \leftarrow a + k_1\ mod\ N$
- $Print$"$\ Choose\ a\ positive\ integer\ less\ than\ \varphi(N)\ to\ be\ your\ exponent$: "
- $Read\ e$
- $e_1 \leftarrow e + k_2 mod\ N$
- $Print$"$\ Enter\ a\ numerical\ value\ \ Message\ to\ be\ encrypted$: "
- $Read\ Message$
- $EncryptedMessage\ \leftarrow Message\ \times a^e\ mod\ N$
- $Print$ "*Encrypted Message is*" :, $EncryptedMessage$
- $Print$"$\ Decryption\ process$!!!!!!! "
- $b \leftarrow a_1 - k_1\ mod\ N$
- $d \leftarrow \varphi(N) - (e_1 - k_1) mod\ N$
- $Print$"$\ Enter\ a\ message\ to\ be\ decrypted$: "
- $Read\ \ EncryptedMessage$
- $DecryptedMessage \leftarrow EncryptedMessage \times b^d mod\ N$
- $Print$ " *Decrypted Message is: "* , *Decrypted Message*
- End

E. Time Complexity

To analyze the time complexity of the above code, we break it into parts:

➢ *GCD Function:*

This function computes the greatest common divisor (GCD) of two integers using the Euclidean algorithm. Its time complexity is typically $O(\log(\min(p, q)))$ where p and q are the input integers.

➢ *Modular Exponentiation Function:*

This function calculates the modular exponentiation of a base raised to an exponent modulo a given modulus. The time complexity of this function is $O(\log(exp))$, where exp is the exponent.

➢ *Main Function:*

- The loop that computes the count of numbers coprime to pn runs from 1 to pn. Inside the loop, the gcd function is called. Therefore, this loop contributes $O(pn \times \log(pn))$ to the time complexity.
- The loop that searches for possible bases runs from 3 to $sqrt(pn)$. Inside the loop, the gcd, function is called. Therefore, this loop contributes $O(sqrt(pn) \times \log(pn))$ to the time complexity.
- The overall time complexity of the main function depends on the dominant terms. Since $O(pn \times \log(pn))$ dominates over $O(sqrt(pn) \times \log(pn))$, the overall time complexity of the main function can be approximated as $O(pn \times \log(pn))$.

Therefore, the time complexity of the code is approximately $O(pn \times \log(pn))$.

F. Analysis of the Algorithm

The results of the algorithm involve the successful encryption and decryption of numerical messages using the implemented public key cryptographic system. Here are the key outcomes:

➢ *Encryption Process:*

The algorithm effectively encrypts a numerical message input by the user using modular arithmetic operations. The encryption process utilizes the selected base a, along with the computed private exponent e and modulus n, to transform the original message into an encrypted form.

➢ *Decryption Process:*

Following encryption, the algorithm decrypts the encrypted message using the computed private exponent d, along with the same base and modulus. This decryption process accurately completes the encryption operation to make the product of encryption key and decryption key, 1. This ensures the original numerical message is recovered without loss or alteration.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY675

➢ *Message Integrity:*

The decrypted message matches the original input message, indicating the integrity and reliability of the encryption and decryption processes. This outcome demonstrates the algorithm's ability to preserve the confidentiality and authenticity of transmitted messages.

➢ *Efficiency and Performance:*

The algorithm exhibits efficient performance in terms of computational complexity and execution time. Modular exponentiation and gcd calculations are performed effectively, allowing for rapid encryption and decryption of messages even for large input values.

➢ *Security Guarantees:*

The algorithm provides security guarantees against common cryptographic attacks due to its reliance on modular arithmetic and the selection of coprime bases. By using these mathematical properties, the algorithm offers resistance to brute force attacks and other cryptographic vulnerabilities. In facts, like ElGamal cryptosystem's security is based on the difficulty of the discrete logarithm problem, also the security of the presented algorithm relies on discrete logarithm problem with in additional that the brute force attacker doesn't know both the base and exponent.

V. CONCLUSION AND RECOMMENDATIONS

In conclusion, the algorithm presented in this study offers a practical and efficient solution for secure communication through encryption and decryption. By using modular exponentiation and the greatest common divisor function, the algorithm facilitates the generation of secure cryptographic keys and the transformation of numerical messages into encrypted forms.

Through a step-by-step process, the algorithm enables users to encrypt sensitive information with confidence, knowing that their data remains confidential and secure during transmission. The integration of modular arithmetic ensures computational efficiency, while the selection of coprime bases enhances the algorithm's resistance to cryptographic attacks.

The results of testing and evaluation demonstrate the algorithm's effectiveness in preserving message integrity and confidentiality. Encrypted messages ca be accurately decrypted, maintaining the original content's integrity without compromise.

Furthermore, the algorithm's simplicity and ease of implementation make it accessible to a wide range of users, from cryptographic experts to practitioners requiring secure communication solutions. As recommendations, we suggest further research to this study and if possible, this algorithm can be merged with another algorithm like elliptic curve cryptography to enhance more security for users who are familiar to prime number modulus. This should be one of the future studies.

REFERENCES

[1]. A. Buchmann, J., Introduction to cryptography, New York: Springer-Verlag, 2001.
[2]. A. Anderson, J., Number theory with application, Carolina at Sparta: University of South: Upper saddle river, 1997.
[3]. U. Dudley, Elementary Number Theory, New Yolk: W. H. FREEMAN AND COMPANY , May 1978 .
[4]. Bart Goddard, Kenneth H. rosen, Elementary Number Theory and its applications, Bosston: Pearson Education, Inc, 2005.

APPENDIX: C++ CODES OF THE ALGORITHM

```
#include<iostream>
#include<cmath>
using namespace std;

int gcd(int p, int q) {
if (q == 0)
return p;
else
return gcd(q, p % q);
}
int modularExponentiation(int base, int exp, int mod) {
int result = 1;
base = base % mod;
while (exp > 0) {
if (exp % 2 == 1)
result = (result * base) % mod;
base = (base * base) % mod;
exp = exp / 2;
}
return result;
}
int main() {
int pn, k_1,k_2, a, a_1, e,e_1, d, b,e_key,
d_key,Encrypted_Message,
message,Encrypted,Decrypted_Message;
cout<< " enter a positive integer : "<<endl;
cin>>pn;
int count;
for(int i=1; i <pn; i++)
{
if (gcd(pn, i)== 1 )
{
count++;
}}
int kk=count;
cout<<" enter a base shift"<<endl;
cin>>k_1;
cout<<" enter an exponent shift"<<endl;
cin>>k_2;

for(int j=3;  j<sqrt(pn); j++)
{
if (gcd(pn, j)== 1 )

cout<<"Possible base:"<<j<<endl;}
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY675

```
cout<<" Choose an appropriate integer from the list provided to be your base:"<<endl;
cin>>a;
a_1= (a+k_1)%pn;
cout<<" Choose a positive integer less than: "<<kk<< " to be your exponent:"<<endl;
cin>>e;
e_1= (e+k_2)%pn;
cout<<" Symmetric keys: are: "<<k_1<<" , "<<k_2<<"."<<endl;
cout<<" Public keys: are: "<<pn<<" , "<<a_1<<" , "<<e_1<<"."<<endl;
e_key =modularExponentiation(a,e, pn);
cout<<" Encryption key= "<<e_key<<endl;
cout<< " Enter a numerical value to be encrypted: "<<endl;
cin>>message;
Encrypted_Message=(( message*e_key)%pn);
cout<<" Encrypted message is: "<<Encrypted_Message<<endl;
cout<<" Decryption process!!!"<<endl;
b= (a_1 -k_1)%pn;
d= (kk- (e_1 -k_2) )%pn;
d_key=modularExponentiation(b, d, pn);
cout<<" Enter a message to be decrypted:"<<endl;
cin>>Encrypted;
Decrypted_Message=(( Encrypted*d_key)%pn);
cout<<" Decrypted message is : "<<Decrypted_Message<<endl;
return 0;}
```