

Assignment-III

Name: P.Abhay

Reg.No: 20BCI0017

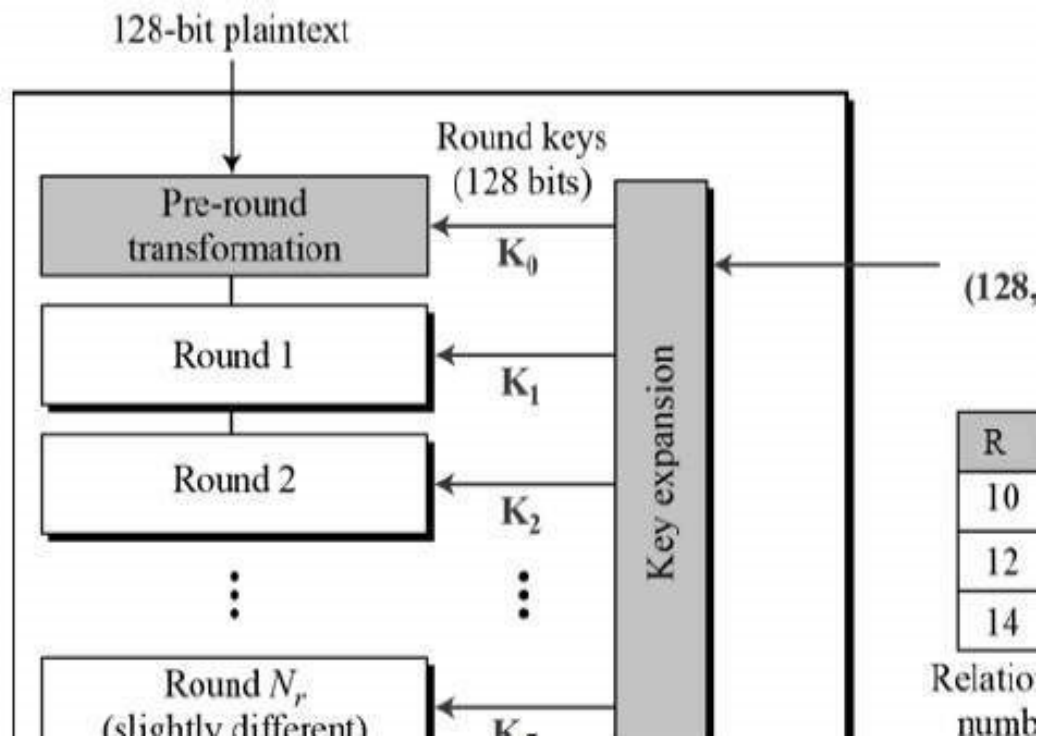
Symmetric Key Algorithm:

AES (Advanced Encryption Standard):

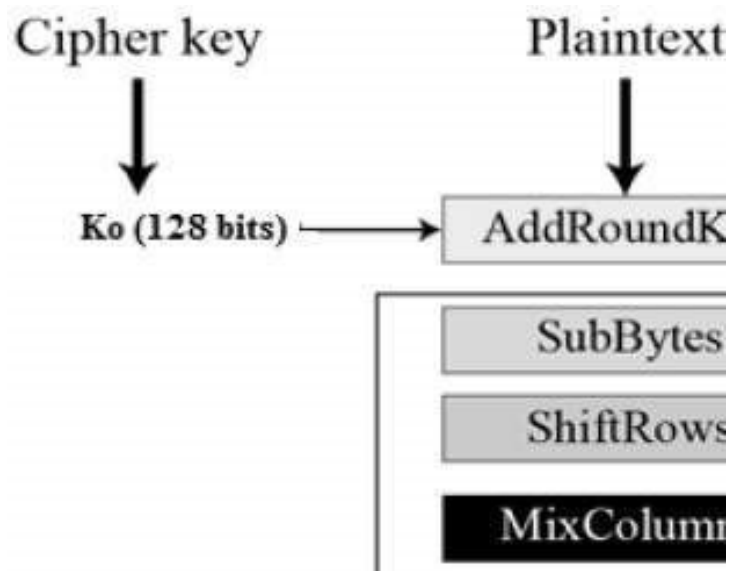
The AES (Advanced Encryption Standard) algorithm is a symmetric encryption algorithm that operates on fixed-size blocks of data. The steps involved in AES are:

1. Key Expansion: AES takes a secret key of 128, 192, or 256 bits and expands it into a set of round keys. The number of rounds depends on the key size, with more rounds for larger keys.
2. Initial Round: AES begins with an initial transformation of the input data using the round key.
3. Rounds: AES performs a series of rounds, each consisting of four main operations: Sub Bytes, Shift Rows, Mix Columns, and Add Round Key. These operations manipulate the data and mix it with the round key.
 - Sub Bytes: Each byte of the data is substituted with a corresponding byte from a fixed substitution table called the S-box.
 - Shift Rows: Bytes in each row of the data matrix are shifted cyclically to the left.
 - Mix Columns: Each column of the data matrix is transformed using a linear mixing operation.
 - Add Round Key: The round key is XORed with the data.
4. Final Round: The final round omits the Mix Columns operation.
5. Output: After the final round, the resulting data is the encrypted ciphertext.

The decryption process for AES is essentially the reverse of the encryption process. The ciphertext goes through a series of inverse operations, including the inverse of SubBytes, ShiftRows, Mix Columns, and Add Round Key, to obtain the original plaintext using the same round keys in reverse order.



IMG: Overall structure of AES [1]



IMG: Process of each round in AES [1]

Strengths of AES:

1. Security: AES is highly secure, resistant to cryptographic attacks when implemented properly.
2. Efficiency: It is computationally efficient, providing fast encryption/decryption.
3. Strong Encryption: AES supports key sizes of 128, 192, and 256 bits, offering robust security.
4. Standardization: Widely adopted and standardized, ensuring compatibility and interoperability.
5. Versatility: AES accommodates different encryption modes for various applications.
6. Hardware Support: It benefits from hardware acceleration and dedicated AES instructions.
7. Publicly Available: AES specifications are openly accessible, encouraging transparency and analysis.

These advantages make AES suitable for secure communication, file encryption, database security, wireless networks, financial systems, and more.

Vulnerabilities and Weaknesses:

1. Side-Channel Attacks: AES implementations may be vulnerable to attacks that exploit physical information leakage.
2. Key Management: Weak or poorly managed encryption keys can compromise AES security.
3. Brute-Force Attacks: In theory, exhaustive key search is possible, but AES's large key sizes make it computationally infeasible.
4. Cryptanalysis: While no practical attacks have been discovered, ongoing research explores potential vulnerabilities.

Real-world Examples of AES:

1. Secure Communication: AES is widely used in secure communication protocols such as SSL/TLS, SSH, and IPsec to encrypt data transmitted over networks.
2. File and Disk Encryption: AES is employed in file encryption tools and disk encryption systems to protect sensitive files and data stored on hard drives, USB drives, and other storage devices.
3. Database Encryption: AES is utilized for encrypting sensitive data stored in databases, adding an additional layer of security to prevent unauthorized access.

4. Wireless Security: AES is a part of the Wi-Fi Protected Access 2 (WPA2) protocol, ensuring secure wireless communication between devices.
5. Financial Systems: AES is often utilized in financial systems to secure transactions and protect sensitive financial information.

Asymmetric Key Algorithm:

RSA Algorithm:

The RSA algorithm is a widely used public-key encryption algorithm that allows secure communication over an insecure network. Here are the steps of RSA:

1. Key Generation:

- Select two large prime numbers, p and q .
- Compute their product, $n = p * q$, which forms the modulus for the RSA algorithm.
- Compute Euler's totient function, $\phi(n) = (p - 1) * (q - 1)$, which represents the number of positive integers less than n that are co-prime to n .
- Choose a public exponent e , which is relatively prime to $\phi(n)$ (i.e., $\gcd(e, \phi(n)) = 1$).
- Compute the private exponent d , such that $(d * e) \bmod \phi(n) = 1$.

2. Encryption:

- To encrypt a message M , convert it into a numerical representation (e.g., using ASCII or Unicode).
- Compute the ciphertext $C = M^e \bmod n$, where $^$ denotes exponentiation.

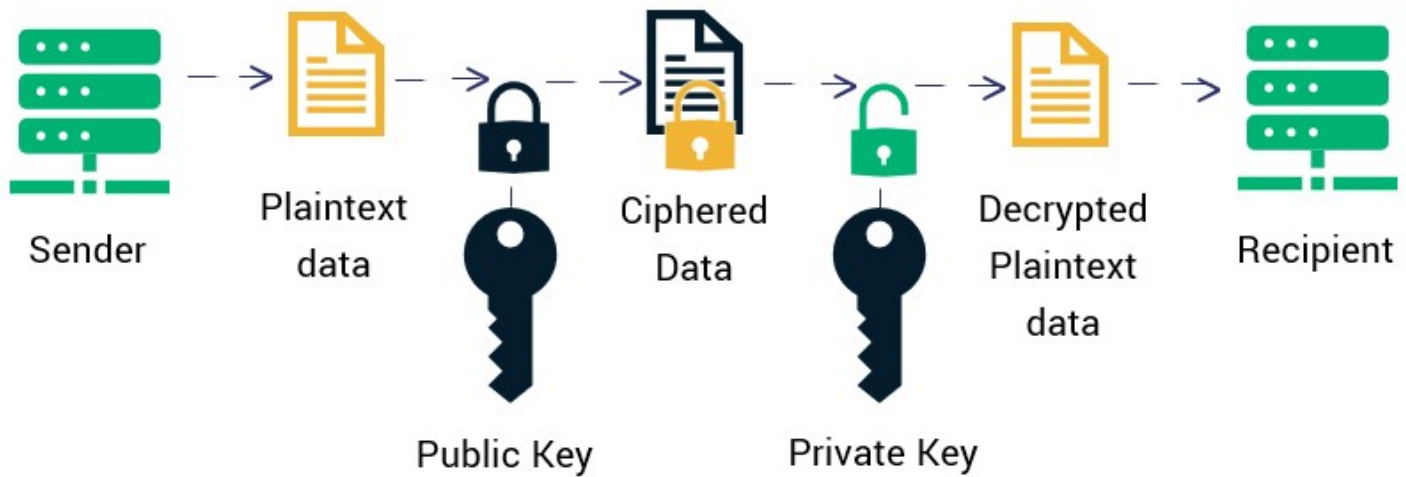
3. Decryption:

- To decrypt the ciphertext C , compute the original message $M = C^d \bmod n$.

The security of RSA relies on the difficulty of factoring large composite numbers into their prime factors. The private key (d, n) is kept secret, while the public key (e, n) is shared openly. Anyone can use the public key to encrypt messages, but only the holder of the private key can decrypt them.

The RSA algorithm provides confidentiality and authenticity. It ensures that the message can only be decrypted by the intended recipient who possesses the private key. Also, RSA can be used for digital signatures, where the sender signs a message with their private key, and the recipient verifies the signature using the corresponding public key.

How RSA Encryption Works



IMG: Working of RSA [3]

Strengths of RSA:

1. **Public-key encryption:** Utilizes separate keys for encryption and decryption, enabling secure communication without a secure key exchange mechanism.
2. **Security:** Based on the computational difficulty of factoring large numbers, offering strong protection for sensitive data.
3. **Digital signatures:** Provides data integrity and sender verification, ensuring message authenticity and non-repudiation.
4. **Flexibility:** Supports various key sizes, allowing users to choose the appropriate level of security for their needs.
5. **Widely adopted:** Standardized and extensively studied, with wide support in cryptographic libraries.
6. **Key management:** Simplifies secure key distribution and access control, ensuring authorized individuals can decrypt and sign messages.

Vulnerabilities and Weaknesses:

1. **Key size:** Insufficiently large key sizes may be vulnerable to attacks as computational power increases, requiring constant evaluation and adjustment.
2. **Quantum computing:** The advancement of quantum computers poses a potential threat to RSA's security, as they can efficiently solve the factorization problem.

3. Timing attacks: RSA implementations need to carefully address timing analysis vulnerabilities by implementing countermeasures to prevent information leakage through timing variations.
4. Side-channel attacks: Protecting against side-channel attacks is crucial, as adversaries can exploit information leaked through power consumption or electromagnetic emissions during RSA operations.
5. Poor random number generation: Ensuring strong random number generation during key generation and encryption is essential to avoid predictable values that could compromise the security of RSA.

Real-world Examples of RSA:

1. Secure Online Communication: RSA is widely used in secure online communication protocols like SSL/TLS. When you access a website over HTTPS, RSA is used for key exchange, establishing a secure connection between your web browser and the server.
2. Digital Signatures: RSA is employed in digital signature schemes used for authentication and data integrity verification. For example, when you digitally sign a document, RSA is used to generate the signature, ensuring that the document remains unaltered and verifying your identity as the signer.
3. Secure Email Communication: RSA is used in email encryption protocols like OpenPGP and S/MIME. It allows users to send encrypted emails, ensuring that the content remains confidential and can only be decrypted by the intended recipients who possess the corresponding private keys.
4. Virtual Private Networks (VPNs): RSA is utilized in VPN technologies for secure remote access. It plays a role in establishing secure tunnels and exchanging encryption keys between the client and the VPN server, ensuring the confidentiality of data transmitted over the network.
5. Secure Shell (SSH) Protocol: RSA is used in the SSH protocol for secure remote access to servers and network devices. It enables secure authentication and encryption of data exchanged between the client and server, preventing unauthorized access and protecting data privacy.

Hash Function:

SHA-256:

SHA-256 (Secure Hash Algorithm 256-bit) is a widely used cryptographic hash function that belongs to the SHA-2 (Secure Hash Algorithm 2) family. Here is how SHA-256 works:

1. Input Message:

- The SHA-256 algorithm takes an input message as a byte stream. The message can be of any length.

2. Padding:

- The input message is padded to ensure that its length is a multiple of 512 bits, as required by the SHA-256 algorithm.

3. Initialization:

- SHA-256 initializes a set of constant values (known as "initial hash values") and a specific message schedule.

4. Processing Blocks:

- The input message is divided into fixed-size blocks (512 bits each) for processing.
- Each block undergoes a series of transformations, including bitwise operations, logical functions, and modular additions.

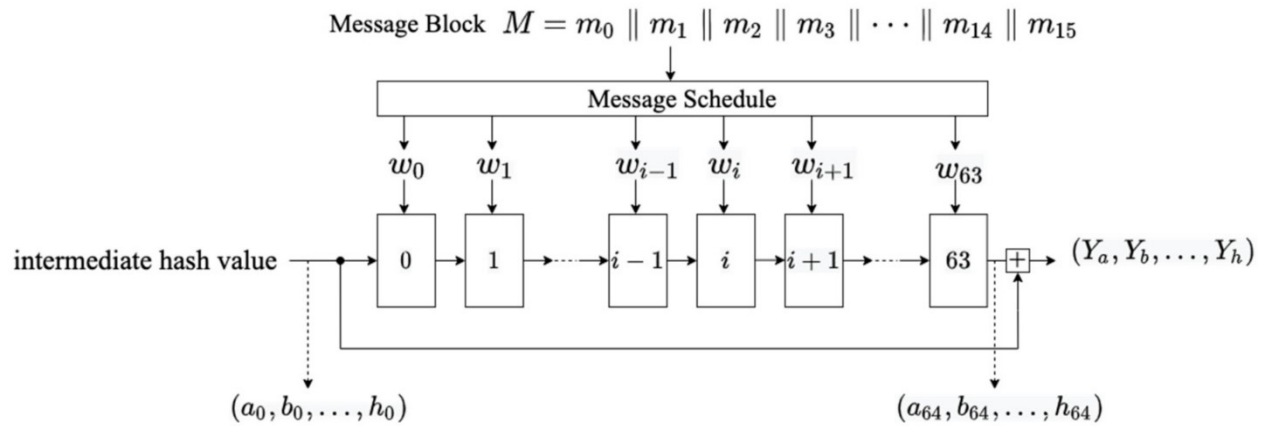
5. Compression Function:

- The compression function operates on each block, combining the current block's data with the result from the previous block.
- This process iterates through all the blocks until the final block is processed.

6. Final Hash Value:

- Once all the blocks are processed, the final hash value is obtained. It is a 256-bit (32-byte) output.

SHA-256 generates a unique hash value for each unique input message, making it suitable for data integrity verification. Even a slight change in the input message will produce a completely different hash output, providing assurance that the data has not been tampered with.



IMG: Architecture of SHA-256 [4]

Strengths of SHA-256:

1. **Strong Cryptographic Security:** SHA-256 is designed to provide a high level of security. It is resistant to pre-image attacks, meaning it is computationally infeasible to determine the original input message from its hash value.
2. **Collision Resistance:** SHA-256 has a large hash space, with a 256-bit output, which significantly reduces the probability of two different inputs producing the same hash value. This makes it highly collision-resistant, ensuring that different data will have different hash values.
3. **Wide Adoption and Standardization:** SHA-256 is widely adopted and standardized, making it a trusted and widely supported cryptographic hash function. It is implemented in various programming languages, cryptographic libraries, and security protocols.
4. **Efficiency:** SHA-256 offers a good balance between security and efficiency. While it is computationally intensive, it can be efficiently implemented on modern computing platforms, providing fast and reliable hashing performance.
5. **Versatility:** SHA-256 is suitable for a wide range of cryptographic applications. It can be used for data integrity checks, password storage, digital signatures, and in blockchain technologies like Bitcoin and other cryptocurrencies.

Vulnerabilities and Weaknesses:

1. **Length Extension Attacks:** While SHA-256 is resistant to some types of attacks, it is vulnerable to length extension attacks. An attacker who knows the hash output and part of the original message can calculate the hash of an extended message without knowing the full content. This can lead to potential security issues in certain scenarios.

2. Collision Attacks: Although SHA-256 has a large hash space, collisions (two different inputs producing the same hash value) are theoretically possible due to the birthday paradox. However, finding actual collisions is computationally infeasible with current technology.
3. Cryptanalysis Advances: As technology advances and new cryptanalytic techniques emerge, vulnerabilities in SHA-256 may be discovered. While no practical attacks against SHA-256 are currently known, it is always important to stay updated with the latest research and recommendations.
4. Quantum Computing: The development of quantum computers has the potential to break the security of SHA-256. Quantum algorithms like Grover's algorithm can significantly reduce the time required to find collisions or pre-images, rendering SHA-256 vulnerable to attacks if a sufficiently powerful quantum computer is developed.
5. Dependence on Hash Length: While SHA-256 provides a 256-bit hash output, the security strength depends on the actual length of the hash used in a specific application. If a shorter hash length is used, the security level may be reduced.
6. Data Input Quality: SHA-256 assumes that the input data is uniformly distributed and random. If the input data exhibits patterns or biases, it may lead to potential vulnerabilities and exploitable weaknesses.

Real-world Examples of SHA-256:

1. Blockchain Technology: SHA-256 is extensively used in blockchain networks like Bitcoin and Ethereum. It is employed in the mining process to secure and validate transactions, create proof-of-work, and maintain the integrity of the blockchain.
2. Digital Signatures: SHA-256 is commonly used in digital signature algorithms like RSA with SHA-256. It ensures the integrity and authenticity of digitally signed documents, verifying that the content has not been tampered with.
3. Password Storage: SHA-256 is employed in password hashing algorithms, such as bcrypt and PBKDF2. It securely stores user passwords by generating a hash value that is stored in the database. During authentication, the entered password is hashed using SHA-256, and the resulting hash is compared to the stored value.
4. SSL/TLS Certificates: SHA-256 is used in the creation and verification of SSL/TLS certificates. It is part of the digital signature process to ensure the authenticity and integrity of the certificates used for secure communication between web servers and clients.

5. File Integrity Checking: SHA-256 can be used to verify the integrity of files by generating a hash value for the file and comparing it with the previously computed hash. If the hash values match, it indicates that the file has not been modified.
6. Software Distribution: SHA-256 hashes are often provided alongside software downloads. Users can verify the integrity of downloaded files by comparing the computed SHA-256 hash with the one provided by the software provider. This ensures that the software has not been tampered with during transmission.

Implementation of RSA:

Problem Statement: Secure Communication between Alice and Bob

Scenario: Alice wants to send a confidential email to Bob. They need a secure communication method that ensures the privacy and integrity of their messages.

Solution: RSA Encryption

1. Key Generation:
 - Alice generates a public-private key pair using large prime numbers.
 - She keeps her private key secret and shares the public key with Bob.
2. Encryption (Sender - Alice):
 - Alice writes her email message and converts it to plaintext.
 - She encrypts the plaintext using Bob's public key, ensuring only Bob can decrypt it.
 - The encrypted message is sent to Bob.
3. Decryption (Receiver - Bob):
 - Bob receives the encrypted message from Alice.
 - He uses his private key to decrypt the ciphertext and obtain the original plaintext.
 - Only Bob, with the possession of his private key, can decrypt the message.
4. Secure Communication:
 - Alice and Bob can exchange confidential emails securely, knowing that the content remains private and unaltered.
 - The encryption process guarantees that only Bob, with his private key, can read the message.

Implementation:

RSA.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
int main()
{
    printf("Enter Prime Number (p): ");
    scanf("%ld", &p);
    flag=prime(p);
    if(flag==0)
    {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("Enter Prime Number (q): ");
    scanf("%ld",&q);
    flag=prime(q);
    if(flag==0||p==q)
    {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("Enter Message: ");
    scanf("%[^\\n]s",msg);
    for(i=0;msg[i]!='\\0';i++)
        m[i]=msg[i];
    n=p*q;
    t=(p-1)*(q-1);
    ce();
    encrypt();
    decrypt();
    return 0;
}
int prime(long int pr)
{
    int i;
    j=sqrt(pr);
    for(i=2;i<=j;i++)
    {
        if(pr%i==0)
            return 0;
    }
    return 1;
}
```

```

void ce()
{
    int k=0;
    for (i=2;i<t;i++)
    {
        if(t%i==0)
            continue;
        flag=prime(i);
        if(flag==1&&i!=p&&i!=q)
        {
            e[k]=i;
            flag=cd(e[k]);
            if (flag>0)
            {
                d[k]=flag;
                k++;
            }
            if(k==99)
                break;
        }
    }
}

long int cd(long int x)
{
    long int k=1;
    while (1)
    {
        k=k+t;
        if (k%x==0)
            return(k/x);
    }
}

void encrypt()
{
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    printf("The Encrypted Message is: ");
    for(i=0;i<len;i++)
    {
        if(m[i]==' ')
        {
            en[i]=' ';
            printf("%c",en[i]);
            continue;
        }
        pt=m[i]-96;
        ct=1;
        for(j=0;j<key;j++)
            ct=(ct*pt)%n;
        en[i]=ct+96;
        printf("%c",en[i]);
    }
}

void decrypt()

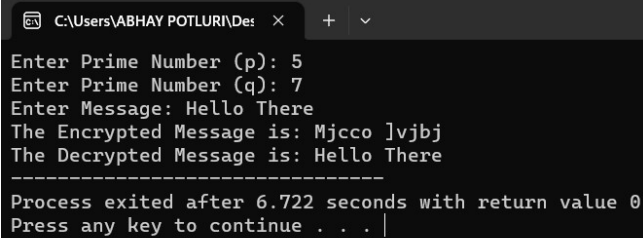
```

```

{
    long int pt,ct,key=d[0],k;
    i=0;
    printf("\nThe Decrypted Message is: ");
    while(en[i]!='\0')
    {
        if (en[i]==' ')
        {
            m[i]=' ';
            printf("%c",m[i]);
            i++;
            continue;
        }
        ct=en[i]-96;
        k=1;
        for(j=0;j<key;j++)
            k=(k*ct)%n;
        pt=k+96;
        m[i]=pt;
        printf("%c",m[i]);
        i++;
    }
}

```

Output:



```

C:\Users\ABHAY POTLURI\Desktop
Enter Prime Number (p): 5
Enter Prime Number (q): 7
Enter Message: Hello There
The Encrypted Message is: Mjcco ]vjbj
The Decrypted Message is: Hello There
-----
Process exited after 6.722 seconds with return value 0
Press any key to continue . . .

```

Security Analysis of RSA:

Potential Threats or Vulnerabilities:

- **Brute Force Attacks:** The prime number selection is critical for the security of RSA. If small or weak primes are chosen, it could make the system vulnerable to brute force attacks.
- **Timing Attacks:** The code implementation does not have any countermeasures against timing attacks, which could potentially leak information about the private key through side-channel analysis.
- **Insecure User Input Handling:** The code lacks input validation, making it susceptible to buffer overflow or input manipulation attacks.
- **Weak Key Generation:** The key generation process could be optimized to ensure a sufficient key length and randomness.

Countermeasures and Best Practices:

- **Secure Prime Number Generation:** Use a secure random number generator to generate large prime numbers.
- **Input Validation:** Implement robust input validation to prevent buffer overflow or malicious input.
- **Timing Attack Mitigation:** Implement countermeasures such as constant-time operations or random delays to prevent timing attacks.
- **Key Length:** Ensure that the key length used in RSA is appropriate for the desired level of security (e.g., 2048 bits or higher).
- **Key Management:** Protect the private key and ensure secure key storage and transmission.

Limitations and Trade-offs:

- The provided code is a basic implementation for educational purposes and may not meet the requirements of a production-ready cryptographic system.
- It does not handle advanced security features like padding schemes or digital signatures, which are essential in real-world RSA implementations.
- The code lacks error handling and may not provide detailed feedback in case of incorrect inputs or unexpected errors.

References:

- [1] [Advanced Encryption Standard \(tutorialspoint.com\)](https://www.tutorialspoint.com/advanced-encryption-standard/aes-algorithm.htm)
- [2] [Advanced Encryption Standard \(AES\) | NIST](https://nist.gov/encryption/advanced-encryption-standard-aes)
- [3] [ECDSA vs RSA: Everything You Need to Know \(sectigostore.com\)](https://sectigostore.com/ecdsa-vs-rsa-everything-you-need-to-know/)
- [4] [Information | Free Full-Text | Algebraic Fault Analysis of SHA-256 Compression Function and Its Application \(mdpi.com\)](https://www.mdpi.com/2076-3413/12/1/1)
- [5] [RSA Algorithm in Cryptography - GeeksforGeeks](https://www.geeksforgeeks.org/rsa-algorithm-cryptography/)
- [6] [What Is SHA-256 Algorithm: How it Works and Applications \[2022 Edition\] | Simplilearn](https://www.simplilearn.com/what-is-sha-256-algorithm-how-it-works-and-applications-2022-edition-2824963.php)