WEEK ASSESSMENT - 3

-By
Prasanna Kumar N
20BCE2121
Branch: CSE(Core)
Vellore Institute of Technology, Vellore Campus

Assignment: Cryptography Analysis and Implementation

Objective:

The objective of this assignment is to analyze cryptographic algorithms and implement them in a practical scenario.

Instructions:

Research: Begin by conducting research on different cryptographic algorithms such as symmetric key algorithms (e.g., AES, DES), asymmetric key algorithms (e.g., RSA, Elliptic Curve Cryptography), and hash functions (e.g., MD5, SHA-256). Understand their properties, strengths, weaknesses, and common use cases.

Analysis:

Here is an analysis of three cryptographic algorithms: the SHA-256 hash function, the RSA symmetric key algorithm, and the AES symmetric key algorithm.

Advanced Encryption Standard (AES):

Brief Explanation:

AES is a symmetric key algorithm that works with data blocks of a fixed size. With multiple rounds of substitution, permutation, and mixing operations, it employs a substitution-permutation network (SPN) structure. Three key sizes are supported by AES: 128 bits, 192 bits, and 256 bits.

Key Strengths and Advantages:

AES is widely used and regarded as secure for many applications. The high level of security it offers is a result of its resistance to well-known cryptographic attacks. AES supports parallel processing and is effective in both hardware and software implementations.

Known Vulnerabilities or Weaknesses:

One common vulnerability in AES encryption implementations is the use of weak keys or IVs. Weak IVs can produce predictable ciphertexts, which attackers can exploit, and weak keys can make the encryption susceptible to attacks. Developers should take care to use robust, random keys and IVs and steer clear of reusing them across various sessions or applications.

Attacks through side-channels are another potential weakness in AES encryption implementations.

Real-World Examples:

AES is frequently used in many different applications, such as protecting communications in VPNs, securing data stored in databases, and protecting sensitive information in financial transactions.

Rivest-Shamir-Adleman (RSA):

Brief Explanation:

In order to exchange keys, create digital signatures, and encrypt and decrypt data, RSA is an asymmetric key algorithm. It is based on how difficult it is computationally to factor large composite numbers into their prime factors. With RSA, a public key and private key pair are created, and the public key and private key are used for encryption and decryption, respectively.

Key Strengths and Advantages:

RSA provides digital signatures and secure key exchanges, enabling secure communication and authentication between parties. Based on the difficulty of factoring large numbers, it is computationally secure. Although RSA keys can be used for both encryption and decryption, symmetric key algorithms are typically faster for large data sets.

Known Vulnerabilities or Weaknesses:

If the keys are not generated correctly or if weak or small key sizes are employed, RSA may be open to attacks. Risks can come from attacks like factorization attacks (using quantum computers, for example) or timing attacks on particular implementations.

Real-World Examples:

RSA is widely used in many protocols and applications, such as PGP (Pretty Good Privacy) for email encryption and digital signatures, SSL/TLS for secure web communication, and SSH for secure remote access.

Secure Hash Algorithm 256-bit (SHA-256):

Brief Explanation:

A common cryptographic hash function is SHA-256, which takes an input message and outputs a fixed-size (256 bits) result. It makes use of a Merkle-Damgrd construction, which iteratively divides and processes the input message into blocks. Since SHA-256 generates a distinct hash value for each distinct input, it can be used to store passwords and verify data integrity.

Key Strengths and Advantages:

Collision resistance is a feature of SHA-256, making it computationally impossible to find two inputs that result in the same hash value. It is a trusted and secure hash function because it is widely used and has undergone extensive analysis.

Known Vulnerabilities or Weaknesses:

Even though SHA-256 is thought to be secure in and of itself, as computing power rises, so does its collision resistance property. The risk of vulnerabilities like length extension attacks makes it crucial to follow the right protocols and procedures in specific scenarios.

Real-World Examples:

Blockchain technology, such as Bitcoin, frequently hashes blocks and guarantees transaction integrity using SHA-256. Additionally, it is used in secure file integrity checking, password hashing (like bcrypt), and digital certificates.

Implementation:

Problem:

The challenge is to implement the RSA algorithm to encrypt the plaintext message with the public key and decrypt it with the private key, thereby ensuring secure communication between the sender and the recipient.

Implementation Steps:

1. Begin by generating p and q, two large prime numbers. The modulus (n) and Euler's totient function (phi) will be calculated using these prime numbers.
2. Determine a public exponent (e) that is relatively prime to phi.
3. Use the modular multiplicative inverse of e modulo phi to compute the private exponent (d).
4. Obtain the user's message in plaintext.
5. Convert the message from plaintext to bytes using UTF-8 encoding.
6. Encrypt the plaintext message using the public key (e, n) and modular exponentiation.
7. Using the private key (d, n) and modular exponentiation, decrypt the ciphertext message.
8. Convert the plaintext message decrypted from bytes to a string.
9. Display the plaintext bytes, the public keys (e, n), the ciphertext, and the deciphered message.

Code Explanation with Code Snippets:

- We begin by importing the required libraries, including random and math for generating random numbers and performing mathematical operations, respectively.

- The is_prime function is used to determine whether a given number is prime.

```python
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return True
```

- Within a given range, two prime numbers (p and q) are generated at random.

```python
p = random.randint(2**31, 2**32)
while not is_prime(p):
    p = random.randint(2**31, 2**32)
```

```python
q = random.randint(2**31, 2**32)
while not is_prime(q) or q == p:
    q = random.randint(2**31, 2**32)
```

- The modulus n is determined by multiplying p and q.

- The totient function (phi) of Euler is computed using (p - 1) and (q - 1).

```
n = p * q
phi = (p - 1) * (q - 1)
```

- The public exponent e is generated arbitrarily within a suitable range, and relative primality is determined using phi.

```
e = random.randint(2, phi)
while math.gcd(e, phi) != 1:
    e = random.randint(2, phi)
```

- Calculating the private exponent d with the modular multiplicative inverse of e modulo phi

```
d = pow(e, -1, phi)
```

- The user is prompted to enter the message in plain text.
- Messages in plaintext are encoded into bytes using UTF-8 encoding.

```
message = input("Enter the plaintext: ")
    plaintext = message.encode('utf-8')
```

- Using the public key (e, n), the plaintext message is converted to an integer and encrypted using modular exponentiation.
- The ciphertext has been acquired.
- Using the private key (d, n) and modular exponentiation, the ciphertext is decrypted.

```
ciphertext = pow(int.from_bytes(plaintext, 'big'), e, n)
decrypted_plaintext = pow(ciphertext, d, n)
```

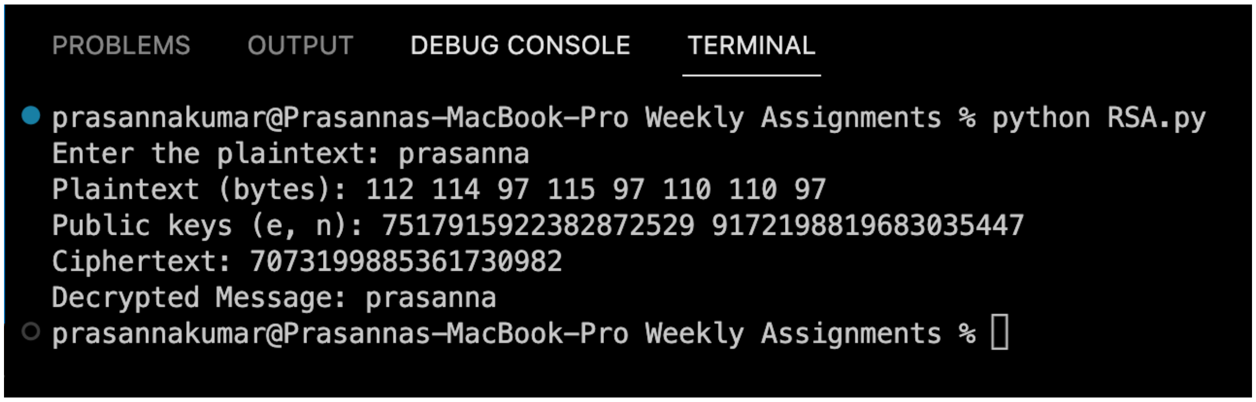
- The decrypted plaintext is converted back to bytes and then UTF-8-decoded into a string. In the event of a UnicodeDecodeError, Latin-1 encoding is used as a replacement.

```
try:
    decrypted_message =
decrypted_plaintext.to_bytes((decrypted_plaintext.bit_length()
+ 7) // 8, 'big').decode('utf-8')
except UnicodeDecodeError:
    decrypted_message =
decrypted_plaintext.to_bytes((decrypted_plaintext.bit_length()
+ 7) // 8, 'big').decode('latin-1')
```

- The terminal displays the plaintext bytes, public keys (e, n), ciphertext, and decrypted message.

```
print("Plaintext (bytes):", " ".join([str(byte) for byte in
plaintext]))
print("Public keys (e, n):", e, n)
print("Ciphertext:", ciphertext)
print("Decrypted Message:", decrypted_message)
```

Output:



```
prasannakumar@Prasannas-MacBook-Pro Weekly Assignments % python RSA.py
Enter the plaintext: prasanna
Plaintext (bytes): 112 114 97 115 97 110 110 97
Public keys (e, n): 7517915922382872529 9172198819683035447
Ciphertext: 7073199885361730982
Decrypted Message: prasanna
prasannakumar@Prasannas-MacBook-Pro Weekly Assignments %
```

Security Analysis:

Potential threats or vulnerabilities:

Brute-Force Attack:

RSA is secure due to the difficulty of factoring large numbers. An adversary could attempt to factorise the modulus n to obtain the prime factors p and q, thereby threatening the algorithm's security.

Chosen Ciphertext Attack:

If an attacker is able to obtain the decrypted plaintext for specific ciphertexts, they may be able to exploit implementation or mathematical weaknesses in RSA.

Weak Random Number Generation:

If the random number generator used to generate prime numbers, the public exponent e, or other critical values is improperly seeded or flawed, it could compromise the implementation's overall security.

Countermeasures or Best Practises:

Use large prime numbers:

Selecting large prime numbers for p and q increases the difficulty of factoring n and strengthens RSA's security.

Secure Random Number Generation:

Implement a secure random number generator that generates prime numbers and other critical values with sufficient entropy.

Key Length and Strength:

Utilise longer key lengths to defend against brute-force attacks. RSA with a 32-bit key size is deemed insecure and should not be employed in practise.

Secure Key Storage:

Protect and securely store the private key to prevent unauthorised access.

Limitations and trade-offs:

Computational Complexity:

Encryption and decryption with RSA require modular exponentiation, which can be computationally expensive, particularly for large key sizes.

Key Distribution:

RSA does not provide a secure key distribution mechanism. To exchange public keys, a secure channel or trusted third party is required.

Padding:

The current implementation employs a straightforward padding strategy. In practise, more secure padding schemes, such as RSA-OAEP, should be employed to prevent certain types of attacks.

Conclusion:

The RSA encryption algorithm was implemented, and its security aspects were analysed in this report. The RSA algorithm is a popular choice for public-key cryptography as it offers a high level of security for data protection and secure communication. It is important to analyse the possible risks and weaknesses that may arise from the implementation of the aforementioned subject matter.

Possible attack vectors were identified, such as brute-force attacks and chosen ciphertext attacks, which have the potential to compromise the security of RSA if not addressed appropriately. The implementation of countermeasures and best practises is crucial to reducing potential risks. This includes the use of large prime numbers, secure random number generation, and the adoption of strong key management practises.

The limitations and trade-offs of implementing RSA were discussed, including the complexity of computation, challenges in distributing keys, and the significance of utilising secure padding schemes.

The significance of cryptography in the fields of cybersecurity and ethical hacking cannot be overstated. The statement highlights the fundamental role of secure communication in ensuring data integrity and confidentiality. Professionals in the cybersecurity field must possess a comprehensive understanding of cryptographic algorithms and their implementation. This knowledge is essential for safeguarding sensitive information, identifying potential vulnerabilities, and ensuring the security of systems and networks.

Through the examination of cryptographic algorithms such as RSA, valuable insights can be gained regarding the fundamental principles of encryption. This can enable professionals to create successful defence strategies against potential security risks. The use of cryptography by ethical hackers can be advantageous in detecting vulnerabilities within cryptographic systems. This can be achieved through the implementation of penetration testing, ultimately leading to the enhancement of security measures for both systems and networks.
Cryptography is a crucial component of cybersecurity and ethical hacking, providing a strong foundation for maintaining confidentiality, securing communication, and safeguarding data and system integrity in today's digital world.