Assignment: Cryptography Analysis and Implementation

Name – Sarthak Verma

Reg no- 20BCI0324

E-mail – sarthak.verma2020@vitstudent.ac.in

Cryptographic Algorithm Analysis:

1. Symmetric Key Algorithm - Advanced Encryption Standard (AES):
   - AES is a symmetric key algorithm widely used for encryption and decryption of electronic data. It operates on fixed-size blocks (128 bits) and supports key sizes of 128, 192, and 256 bits.
   - How it works: AES employs a substitution-permutation network (SPN) structure. It involves several rounds of substitution, permutation, and mixing operations using a series of key-dependent transformations.
   - Key strengths and advantages: AES is considered highly secure and resistant to known attacks. It provides a high level of confidentiality and is computationally efficient on modern hardware.
   - Vulnerabilities or weaknesses: As of my knowledge cutoff in September 2021, no practical vulnerabilities have been found in the AES algorithm.
   - Real-world examples: AES is commonly used to secure sensitive data in various applications, including online banking, VPNs, wireless networks (WPA2), and file encryption.
2. Asymmetric Key Algorithm - RSA:
   - RSA is an asymmetric key algorithm widely used for secure communication, digital signatures, and key exchange. It relies on the difficulty of factoring large composite numbers.

- How it works: RSA involves generating a public-private key pair. The public key is used for encryption, while the private key is used for decryption or signing. It relies on the mathematical properties of modular exponentiation and the difficulty of factoring large numbers.
- Key strengths and advantages: RSA provides a secure method for key exchange and digital signatures. It is computationally expensive, but its security is based on the difficulty of factoring large numbers.
- Vulnerabilities or weaknesses: RSA is vulnerable to attacks if the key size is insufficient. Attacks such as factorization and timing attacks can be used against weak RSA implementations.
- Real-world examples: RSA is commonly used in secure email (PGP/GPG), SSL/TLS for secure web browsing, secure file transfer (SFTP), and digital certificate issuance.

3. Hash Function - SHA-256:
    - SHA-256 is a cryptographic hash function that takes an input and produces a fixed-size output (256 bits). It is commonly used for data integrity verification and password storage.
    - How it works: SHA-256 operates by performing a series of bitwise logical operations and modular additions on the input message. It generates a unique hash value that is highly unlikely to produce the same output for different inputs.
    - Key strengths and advantages: SHA-256 is considered highly secure and resistant to collision attacks. It is fast, produces fixed-length outputs, and any slight change in the input will result in a significantly different hash value.
    - Vulnerabilities or weaknesses: While SHA-256 itself is still secure, the main vulnerability lies in the potential for collisions due to the limited size of the output space. As computing power increases, the likelihood of finding collisions increases.
    - Real-world examples: SHA-256 is commonly used for password storage (in combination with techniques like salting), digital signatures, blockchain technology (e.g., Bitcoin), and data integrity checks.

Now that we have analyzed the algorithms, let's move on to the implementation of one of them in a practical scenario.
Implementation:
Scenario: Secure File Encryption using AES
Implementation Steps:

4. Choose a programming language that supports AES encryption. Python is a popular choice.
5. Install the required cryptographic library. In Python, you can use the **cryptography** library.
6. Generate a random AES key of the desired key size (128, 192, or 256 bits).
7. Read the input file and convert it to binary or text format, depending on the requirements.
8. Encrypt the input file using AES and the generated key. Apply appropriate padding if needed.
9. Save the encrypted file to a specified location.
10. To decrypt the file, read the encrypted file, apply AES decryption using the same key, and save the decrypted file to the desired location.

Python code snippet for AES encryption:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes

def encrypt_file_AES(key, input_file_path, output_file_path):
    with open(input_file_path, 'rb') as input_file:
        plaintext = input_file.read()

    # Generate random IV (Initialization Vector)
    iv = b'\x00' * 16  # You should use a securely generated IV in practice

    # Pad the plaintext
    padder = padding.PKCS7(128).padder()
    padded_plaintext = padder.update(plaintext) + padder.finalize()

    # Encrypt the padded plaintext
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv),
backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(padded_plaintext) + encryptor.finalize()

    # Write the encrypted ciphertext to the output file
    with open(output_file_path, 'wb') as output_file:
        output_file.write(ciphertext)
```

Security Analysis: To perform a security analysis of this implementation, should consider the following:

11. Key Management: Ensure that the encryption keys are securely generated, stored, and handled. Avoid hardcoding keys in the code.
12. Encryption Mode: The chosen mode of operation (CBC in the example) should be secure against known attacks, such as padding oracle attacks.
13. Initialization Vector (IV): Generate a random IV for each encryption operation to prevent patterns from emerging in the ciphertext.
14. Padding: Apply appropriate padding schemes to ensure proper block alignment and avoid information leakage.
15. Secure File Handling: Implement secure file handling practices to prevent unauthorized access or tampering with encrypted files.
16. Key Exchange: Consider the secure exchange of the encryption key between the sender and receiver if required.

Conclusion: Cryptography plays a crucial role in ensuring the confidentiality, integrity, and authenticity of data in various fields, including cybersecurity and ethical hacking. It provides secure methods for protecting sensitive information, establishing secure communication channels, verifying data integrity, and authenticating entities. Understanding different cryptographic algorithms, their strengths, weaknesses, and proper implementation practices is vital for building secure systems and defending against attacks. By implementing cryptographic algorithms correctly and following recommended security practices, we can enhance the security of our systems and protect sensitive information from unauthorized access or manipulation.