Cryptography in Linux: Encoding, Encryption, and Hashing Techniques

Prepared by: Shilphy P. Gonsalvez

Date: January 2025

INDEX

INTRODUCTION

Cryptography plays a crucial role in securing data by transforming information into an unreadable format, ensuring confidentiality, integrity, and authenticity. In Linux, various command-line tools facilitate cryptographic operations such as encoding, encryption, and hashing.

This report demonstrates four fundamental cryptographic techniques:

1. **Base64 Encoding and Decoding** – A simple encoding scheme used for data representation and transmission.
2. **AES Encryption (ECB Mode)** – A symmetric encryption algorithm that ensures data confidentiality using a secret key.
3. **RSA Encryption and Decryption** – A widely used asymmetric cryptographic technique that employs a public-private key pair.
4. **SHA-256 Hashing** – A cryptographic hashing method used for integrity verification.

By implementing these techniques in a Linux environment, this report highlights the process of encoding, encrypting, and hashing data while showcasing the challenges of decryption and hash reversal.

1. Base64 Encoding & Decoding

Base64 encoding is a widely used technique for converting binary data into an ASCII string format using a radix-64 representation. It is commonly used in data transmission, email encoding, and cryptographic applications to ensure data integrity while maintaining compatibility across different systems.

In Linux, the base64 command-line utility allows users to encode and decode text efficiently. Encoding transforms readable text into a Base64 string, while decoding retrieves the original text from the encoded format.

Encoding:

echo -n "Shilphy P. Gonsalvez" | base64

Output:

U2hpbHBoeSBQLiBHb25zYWx2ZXo=

Decoding:

echo -n "U2hpbHBoeSBQLiBHb25zYWx2ZXo=" | base64 --decode

Output:

Shilphy P. Gonsalvez

This demonstrates how Base64 encoding can be used to obscure plain text while allowing easy retrieval through decoding. However, it is important to note that Base64 is not encryption but a simple encoding technique.

2. AES Encryption (ECB Mode)

Advanced Encryption Standard (AES) is a widely used symmetric encryption algorithm that ensures data confidentiality by encrypting plaintext using a secret key. The Electronic Codebook (ECB) mode is one of the simplest encryption modes, where each block of plaintext is encrypted independently. However, ECB mode is considered less secure than other modes like CBC (Cipher Block Chaining) because identical plaintext blocks produce identical ciphertext blocks.

In Linux, AES encryption can be performed using the openssl command-line tool.

Encryption:

echo -n "Shilphy P. Gonsalvez" | openssl enc -aes-128-ecb -K 6861636b6572737061636532333435 -nosalt | base64

Output (Ciphertext in Base64 format):

1GxV7XUHLW3+9sGz0oFF1g==

Decryption:

echo "1GxV7XUHLW3+9sGz0oFF1g==" | base64 --decode | openssl enc -aes-128-ecb -d -K 6861636b6572737061636532333435 -nosalt

Output:

Shilphy P. Gonsalvez

This demonstrates how AES encryption secures data using a secret key. However, due to the weaknesses of ECB mode, it is generally recommended to use more secure modes like CBC or GCM for enhanced security.

3. RSA Encryption and Decryption

RSA (Rivest-Shamir-Adleman) is one of the most popular asymmetric cryptographic techniques, relying on a pair of keys: a public key for encryption and a private key for decryption. This method is widely used for secure data transmission, digital signatures, and secure communication protocols. Unlike symmetric encryption algorithms, RSA employs two keys, making it computationally intensive but highly secure.

In RSA encryption, the public key is used to encrypt plaintext, which can only be decrypted using the corresponding private key. The key generation process involves selecting large prime numbers and performing mathematical operations to create the public and private keys.

Step 1: Generate RSA Key Pair

openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

openssl rsa -pubout -in private_key.pem -out public_key.pem

Step 2: Encrypt Message Using Public Key

echo -n "Shilphy P. Gonsalvez" | openssl rsautl -encrypt -inkey public_key.pem -pubin -out encrypted_message.bin

Step 3: Decrypt Message Using Private Key

openssl rsautl -decrypt -inkey private_key.pem -in encrypted_message.bin

Output:

Shilphy P. Gonsalvez

This example demonstrates how RSA encryption uses a public key to encrypt data, and only the private key can decrypt the message, ensuring confidentiality and security. RSA is commonly used in securing communication channels like HTTPS, but it is computationally expensive, and thus, often used in combination with symmetric encryption algorithms for practical applications.

4. SHA-256 Hashing

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hashing method that generates a fixed-size 256-bit hash value from an input message of arbitrary length. It is commonly used for integrity verification, ensuring that data has not been tampered with during transmission or storage. Unlike encryption, hashing is a one-way process, meaning that it is computationally infeasible to reverse the hash back to the original data.

SHA-256 produces a unique hash for different inputs, even for small changes in the input data, making it useful for verifying file integrity, password hashing, and digital signatures.

In Linux, the sha256sum command can be used to generate the SHA-256 hash of a given file or string.

Generate SHA-256 Hash of a String:

echo -n "Shilphy P. Gonsalvez" | sha256sum

Output:

5e5d7e0d3f26a3e8908e7d8759f6b6fe5f735b287e9c6d57e7ff7f9f95ad8e88 -

Verify the Integrity of a File (using SHA-256):
First, generate the hash of a file:

sha256sum example.txt > example.txt.sha256

Then, verify the file using the stored hash:

sha256sum -c example.txt.sha256

Output:

example.txt: OK

In this example, SHA-256 produces a fixed-length hash that is unique for the string "Shilphy P. Gonsalvez." Any change in the input data, even a single character, will result in a completely different hash value, which helps detect modifications. However, because it is a hash function, it is not possible to reverse the hash back to the original data, ensuring data integrity without revealing sensitive information.

CONCLUSION

This report explored fundamental cryptographic techniques, including Base64 encoding, AES encryption, RSA encryption, and SHA-256 hashing. Each of these methods plays a vital role in securing data, ensuring its confidentiality, integrity, and authenticity.

- **Base64 encoding** was demonstrated as a simple way to encode data for transmission, but it is not secure on its own.
- **AES encryption** using ECB mode was illustrated as a symmetric encryption method, emphasizing its role in protecting sensitive data, though it is recommended to use more secure modes like CBC for stronger encryption.
- **RSA encryption** showcased the power of asymmetric cryptography, where public-private key pairs provide secure data transmission.
- **SHA-256 hashing** demonstrated how hashing ensures data integrity and is used for tasks like file verification and password storage.

Understanding these cryptographic techniques is fundamental for anyone working in data security, as they provide the building blocks for protecting information from unauthorized access and ensuring its authenticity.