Universidade Federal do Ceará
Campus Quixadá
QXD0099 - Desenvolvimento de Software para Persistência
Prof. Francisco Victor da Silva Pinheiro

Lista 7 - Encriptação e Decriptação de arquivos e Verificação de Integridade

Objetivo:

Este trabalho envolve a implementação de criptografia e decriptação assimétrica utilizando o algoritmo RSA. O processo é complementado pela verificação de integridade dos arquivos, assegurando que o conteúdo não foi alterado durante as operações.

1. Criptografia e Decriptação Assimétrica com RSA:

- Criptografia: Um arquivo será protegido usando uma chave pública RSA, garantindo que apenas a chave privada correspondente possa reverter o processo.
- Decriptação: O arquivo criptografado será restaurado ao seu estado original utilizando a chave privada.

2. Verificação de Integridade com SHA-256:

- Após a decriptação, o script calcula o hash do arquivo original e do arquivo decriptado.
- Comparação de Hashes: Os valores hash, gerados com o algoritmo SHA-256, serão comparados para confirmar se os dois arquivos possuem conteúdo idêntico.

3. Código base

```
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.asymmetric.padding import OAEP,
MGF1
from cryptography.hazmat.primitives.hashes import SHA256
import hashlib
from pathlib import Path

# Gerar chaves assimétricas
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
```

```
public_key = private_key.public_key()

# Função para criptografar com chave pública
def encrypt_asymmetric(file_path, public_key):
    with open(file_path, 'rb') as f:
        file_data = f.read()
    encrypted_data = public_key.encrypt(
        file_data,
        OAEP(mgf=MGF1(algorithm=SHA256()), algorithm=SHA256(),
label=None)
    )
    encrypted_file = f"{file_path}.enc"
    with open(encrypted_file, 'wb') as f:
        f.write(encrypted_data)
    print(f"Arquivo criptografado: {encrypted_file}")
    return encrypted_file

# Função para decriptar com chave privada
def decrypt_asymmetric(file_path, private_key):
    with open(file_path, 'rb') as f:
        encrypted_data = f.read()
    decrypted_data = private_key.decrypt(
        encrypted_data,
        OAEP(mgf=MGF1(algorithm=SHA256()), algorithm=SHA256(),
label=None)
    )
    decrypted_file = file_path.replace('.enc', '.dec')
    with open(decrypted_file, 'wb') as f:
        f.write(decrypted_data)
    print(f"Arquivo decriptado: {decrypted_file}")
    return decrypted_file

# Função para calcular hash SHA-256
def calculate_sha256(file_path):
    with open(file_path, 'rb') as f:
        file_data = f.read()
    sha256_hash = hashlib.sha256(file_data).hexdigest()
    print(f"SHA-256 do arquivo {file_path}: {sha256_hash}")
    return sha256_hash
```

"A melhor maneira de predizer o futuro é inventá-lo."
Alan Kay