Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 5

Discipline: Cryptography and Security

Elaborated: Berco Andrei, FAF - 221

Checked: asist. univ., Nirca Dumitru

Chișinău 2024

Topic: Theme: Public Key Infrastructure (PKI) and Digital Signature Algorithm (DSA)

Tasks:

Create an internal PKI using the OpenSSL tool. The generation of the root private key and the initialization of a Certified Authority (CA) are required. A self-signed certificate is created for the CA. System must be able to issue and revoke the private key for the user so that he can subsequently generate a digital signature. Each user/entity that obtains a signature will be able to sign the document/file and verify this signature. For the realization of this laboratory, the use of any language is allowed, including script languages such as Bash, PowerShell, zsh etc. Requirements:

● Use algorithm RSA for generating private keys.

● Users' private keys validity is 365 days and dimension keys are minimum 2048 bits.

● Private key dimension for CA is 4096 bits and expired time for self-signed certificate in 10 years (3650 days)

Theoretical notes:

1. Public Key Infrastructure (PKI)

Definition:
PKI is a framework for managing public and private keys, digital certificates, and the Certificate Authorities (CAs) that issue them. It enables secure communication through encryption and digital signatures.

Components:

- **Public and Private Keys**: Asymmetric cryptography where the public key encrypts data and the private key decrypts it.
- **Digital Certificates**: Digital documents that associate a public key with an identity, issued by a trusted Certificate Authority (CA).
- **Certificate Authority (CA)**: A trusted entity that issues and manages digital certificates.
- **Registration Authority (RA)**: Intermediary that authenticates the entity before issuing a certificate.

2. Digital Signature Algorithm (DSA)

Definition:
DSA is a cryptographic algorithm used for digital signatures. It provides integrity and authentication by signing data with a private key, which can be verified by anyone with the corresponding public key.

Working:

- **Signing**: The sender uses their private key to create a signature on a message or document.
- **Verification**: The recipient uses the sender's public key to verify the signature and ensure the data hasn't been altered.

Applications:

- Ensuring message authenticity, integrity, and non-repudiation in digital communications.

Implementation:

```
    # Generate CA private key
    print("Generating CA private key...")
    ca_private_key = rsa.generate_private_key(
        public_exponent=65537, key_size=4096
    )
```

This line generates a private key for the Certificate Authority (CA) using RSA encryption with a key size of 4096 bits.

```
 ca_cert = (
        CertificateBuilder()
        .subject_name(subject)
        .issuer_name(issuer)
        # Public key of the CA
        .public_key(ca_private_key.public_key())
        # Unique serial number for the certificate
        .serial_number(random_serial_number())
        # Validity period of the certificate
        .not_valid_before(datetime.now(timezone.utc))
        # Validity period of the certificate
        .not_valid_after(datetime.now(timezone.utc) + timedelta(days=3650))
        # Basic constraints extension to indicate that this is a CA certificate
        .add_extension(BasicConstraints(ca=True, path_length=None),
critical=True)
        # Sign the certificate with the CA private key
        .sign(private_key=ca_private_key, algorithm=hashes.SHA256())
    )
```

This block generates a self-signed certificate for the CA, using its own private key to sign the certificate.

```
    # Generate user private key
    print(f"Generating private key for user: {username}...")
    user_private_key = rsa.generate_private_key(
        public_exponent=65537, key_size=2048
    )
```

This line generates a private key for the user using RSA encryption with a smaller key size (2048 bits).

```
user_cert = (
        CertificateBuilder()
        .subject_name(subject)
        .issuer_name(ca_cert.subject)
        .public_key(user_private_key.public_key())
        .serial_number(random_serial_number())
        .not_valid_before(datetime.now(timezone.utc))
        .not_valid_after(datetime.now(timezone.utc) + timedelta(days=365))
        .add_extension(BasicConstraints(ca=False, path_length=None),
critical=True)
        .sign(private_key=ca_private_key, algorithm=hashes.SHA256())
    )
```

This block creates a user certificate, signed by the CA's private key, ensuring it is trusted by the CA.

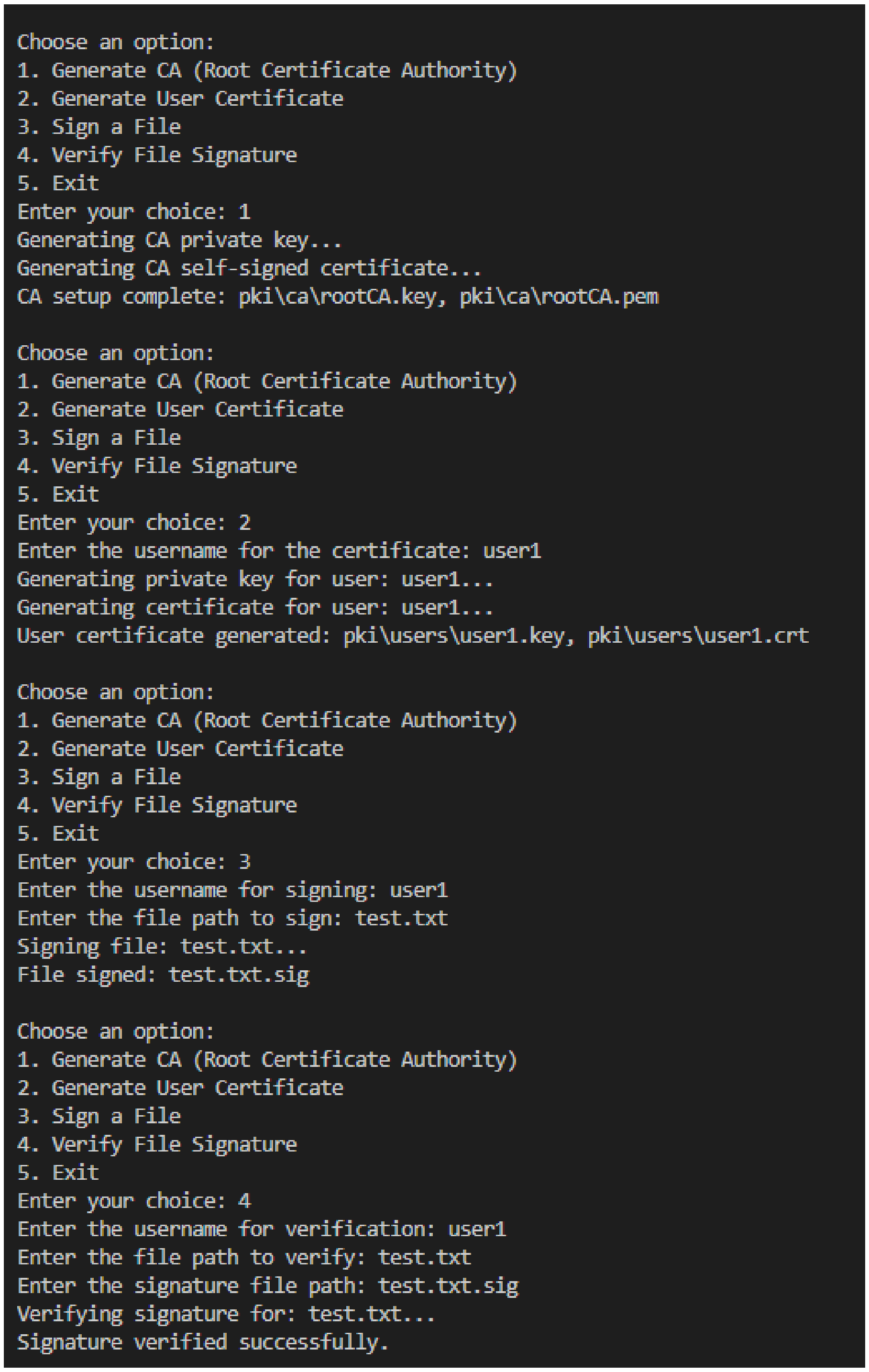
```python
# Sign the file
    print(f"Signing file: {file_path}...")
    signature = user_private_key.sign(
        data,
        # (Probabilistic Signature Scheme) padding scheme for RSA
        padding.PSS(
            # Mask generation function (MGF) used in PSS, which is a key part of the padding scheme
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH,
        ),
        hashes.SHA256(),
    )
```

This signs the data using the user's private key with the PSS padding scheme (Probabilistic Signature Scheme) and SHA-256 hash function.

```python
 print(f"Verifying signature for: {file_path}...")
    try:
        user_public_key.verify(
            signature,
            data,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH,
            ),
            hashes.SHA256(),
        )
        print("Signature verified successfully.")
    except Exception as e:
        print("Verification failed:", e)
```

This verifies the file signature using the user's public key and the PSS padding scheme.

Results:



```
Choose an option:
1. Generate CA (Root Certificate Authority)
2. Generate User Certificate
3. Sign a File
4. Verify File Signature
5. Exit
Enter your choice: 1
Generating CA private key...
Generating CA self-signed certificate...
CA setup complete: pki\ca\rootCA.key, pki\ca\rootCA.pem

Choose an option:
1. Generate CA (Root Certificate Authority)
2. Generate User Certificate
3. Sign a File
4. Verify File Signature
5. Exit
Enter your choice: 2
Enter the username for the certificate: user1
Generating private key for user: user1...
Generating certificate for user: user1...
User certificate generated: pki\users\user1.key, pki\users\user1.crt

Choose an option:
1. Generate CA (Root Certificate Authority)
2. Generate User Certificate
3. Sign a File
4. Verify File Signature
5. Exit
Enter your choice: 3
Enter the username for signing: user1
Enter the file path to sign: test.txt
Signing file: test.txt...
File signed: test.txt.sig

Choose an option:
1. Generate CA (Root Certificate Authority)
2. Generate User Certificate
3. Sign a File
4. Verify File Signature
5. Exit
Enter your choice: 4
Enter the username for verification: user1
Enter the file path to verify: test.txt
Enter the signature file path: test.txt.sig
Verifying signature for: test.txt...
Signature verified successfully.
```

Conclusion:

In this project, we have successfully implemented a simple Public Key Infrastructure (PKI) system that allows the creation of a Certificate Authority (CA), the generation of user certificates signed by the CA, and the ability to sign and verify files using RSA encryption and digital signatures.

1. **Certificate Authority (CA) Setup**:
   We began by creating a root CA with its private key and a self-signed certificate. This CA acts as a trusted entity responsible for signing other certificates, ensuring the authenticity and integrity of digital communications.
2. **User Certificate Generation**:
   Users can generate their own private keys and certificates, which are then signed by the CA. This process ensures that the certificates issued by the CA are trusted, as they can be verified against the CA's public certificate.
3. **File Signing and Verification**:
   We implemented the ability to sign files with a user's private key, ensuring that the file has not been altered and comes from a verified source. The signature can be verified using the user's public key, ensuring the integrity and authenticity of the file.
4. **Security Protocols**:
   Throughout the process, we utilized secure cryptographic algorithms such as RSA for key generation and the PSS padding scheme with SHA-256 for signing and verifying messages. These techniques ensure strong security for digital communications.

Overall, this project demonstrates how public key cryptography, digital certificates, and digital signatures can be used to secure communications and verify the authenticity of both users and files. By implementing these cryptographic protocols, we create a robust system for secure data exchange and authentication.