Lab - Generate and Use a Digital Signature

Objectives

- Use OpenSSL to generate a digital signature.
- Sign a document with the digital signature.
- Verify that a signed document has been changed.

Background / Scenario

A digital signature is a mathematical technique used to validate the authenticity and integrity of a digital message. The purpose of a digital signature is to prevent tampering and impersonation in digital communications. In many countries, including the United States, digital signatures have the same legal significance as traditional forms of signed documents. The United States Government now publishes electronic versions of budgets, laws, and congressional bills with digital signatures.

A digital signature algorithm consists of a signature creation and signature verification process. User A generates the digital signature and User B verifies the signature using the verification process. Both the signer and the verifier have a public and private key that they use to complete each process.

In this lab, you will use the toolkit OpenSSL to generate a digital signature. You will then generate a document, sign it with the digital signature, and then validate the authenticity and integrity of the document. Finally, you will change the document and then validate that the document is no longer authentic because its integrity has been compromised.

Required Resources

PC with the **CSE-LABVM** installed in VirtualBox

Instructions

Step 1: Open a terminal window in the CSE-LABVM.

a. Launch the **CSE-LABVM**.

b. Double-click the **Terminal** icon to open a terminal.

Step 2: Generate and view a private key.

a. To generate a private key, use the **openssl genpkey** command. The command generates a private key using the RSA algorithm and outputs it to a file named **private_key.pem**.

```
cisco@labvm:~$ openssl genpkey -algorithm RSA -out private_key.pem
.........................+++++
...+++++
cisco@labvm:~$
```

b. Use the **cat** command to view the **private_key.pem**.

```
cisco@labvm:~$ cat private_key.pem
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQC1db50XOYeDTAy
GnQLRwGusr7us0Mi44hfFUmai3QHzelqRBxO06ujv9fFwQ8e5QsaQWbph+RVTQBu
<output omitted>
```

```
R7TLUOrewnIlkMuVLk8II2EQAXTMmvvZOICCiTSvm8gflx/FRJmUEiTf0I0MVUai
X6O9rDJOjnoHBbi67+fgN0sn
-----END PRIVATE KEY-----
cisco@labvm:~$
```

Step 3: Generate and view a public key.

a. To generate a public key, use the **openssl pkey** command. The command takes your **private_key.pem** as an input, and then outputs a public key (**-pubout -out**) to a file called **public_key.pem**.

```
cisco@labvm:~$ openssl pkey -in private_key.pem -pubout -out public_key.pem
cisco@labvm:~$
```

b. Use the **cat** command to view the **public_key.pem**.

```
cisco@labvm:~$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtXW+dFzmHg0wMhp0C0cB
rrK+7rNDIuOIXxVJmot0B83pakQcTtOro7/XxcEPHuULGkFm6YfkVU0Abq/1ccub
SFZFb1kKobMutMhxvfBQjQr3jo1j6aG12yMvrQudvSlaxHTvuDpMlhPY3IrBjkMz
2Lx0SjjOC9uD3CzaMwu6JWnhRt0svH7n6cZNXDfIlsYpcjLg9JqKrWEO3Ooq05q1
e2JEAuDOXte+M2O0ZC7cjSyxCiOfD2IEXkBq41H7xgIKTq2WJ8f+/RXEdC5Mx6Xx
fu7pQXN9gT9LbUPlLJfUG7vTCs2d2AA6TBUyUuzH2mS61KwWcT8VRAWtWdr/sPSK
AQIDAQAB
-----END PUBLIC KEY-----
cisco@labvm:~$
```

Step 4: Create a new document that will be digitally signed.

a. Use the **echo** command to create a text file named **contract.txt**.

```
cisco@labvm:~$ echo Please transfer 2,000,000 US Dollars to Mr. Jester by 6pm
today! > contract.txt
cisco@labvm:~$
```

b. Use the **cat** command to view the **contract.txt** file.

```
cisco@labvm:~$ cat contract.txt
Please transfer 2,000,000 US Dollars to Mr. Jester by 6pm today!
```

Step 5: Use the private key to digitally sign the new document.

a. To sign the document, use the **openssl dgst** command. The **dgst** command can take any number of message digest values. In this example, you will use SHA 256, and then use **private_key.pem** to output a **signature** for the **contract.txt** document.

```
cisco@labvm:~$ openssl dgst -sha256 -sign private_key.pem -out signature
contract.txt
cisco@labvm:~$
```

b. Use the **cat** command to view the **signature** file. The file is a binary file. Press **Enter** to get a new command line.

```
cisco@labvm:~$ cat signature
H&/J c M $R xpA *t > ъmr C      jw q]t' #ot"%_B X ~ k p3 - .
<output omitted>
RO      D?Nz f <H ~ P5nJ hqG &28Jcisco@labvm:~$
```

Step 6: Verify the authenticity and integrity of the document.

Digital signature technology allows the recipient to verify the file's authenticity and integrity. The process of digital signature verification is to ensure that a given message has been signed by the private key that corresponds to a given public key.

To verify that the document is authentic and has not been tampered with, use the **openssl dgst** command with the **verify** option and the **public_key.pem**.

```
cisco@labvm:~$ openssl dgst -sha256 -verify public_key.pem -signature
signature contract.txt
Verified OK
```

Step 7: Simulate a threat actor changing the specified recipient in the contract.txt file.

a. Use **gedit** to open the **contract.txt** file.

```
cisco@labvm:~$ gedit contract.txt
```

b. Change **Mr. Jester** to **Mr. Viper**.

c. Click **File** > **Quit**, and then click **Save** in the dialog box.

Step 8: Verify that the document's integrity has been compromised.

Reuse the **openssl dgst** command with the **verify** option to validate that the document's verification now fails.

```
cisco@labvm:~$ openssl dgst -sha256 -verify public_key.pem -signature
signature contract.txt
Verification Failure
cisco@labvm:~$
```