MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

"KHARKIV POLYTECHNICAL INSTITUTE"

Department of Computer Engineering and Programming

«Software Means of Information Protection »

*Laboratory work report No 7*

*Topic: «* *Electronic digital signature»*

Student:

Group. KH 919 i.e.

Okechukwu Chukwuemeka Onyekwere-Dike

Verified by:

Lecturer Viktor CHELAK

Kharkiv – 2022

***Purpose of work:***

Creating a program for generating and verifying messages with an electronic digital signature.

***Individual task:***

• Message signature program.

• Message verification program

Variant 8: Backpack encryption algorithm.

***Algorithm of the program:***

The program has three functionalities:

1. Generate key pairs.
2. Sign document (Create signature for specific data)
3. Verification of signature.

*#Generate key pairs:*

For creation of new key pairs (public, private) to work with in signature

For the public key:

This key has a length of 8 (Since we are going to use it for every byte or 8-bits) we create the initial value randomly, and next values must be greater than the sum of previous values (using random value in range).

For the private key:

We must select a number “m” that is greater the sum of all values of private key, and a number “n” that has no common factor with m. Therefore private key should be: key private (i) = n \* key public (i) mod (m)

Last step for key generation is to save it into a json file and request a path to save the file from user. Also, I’ve implemented a check for selected path if it contains already a key file and ask user to replace it or not.

*#Sign document (data):*

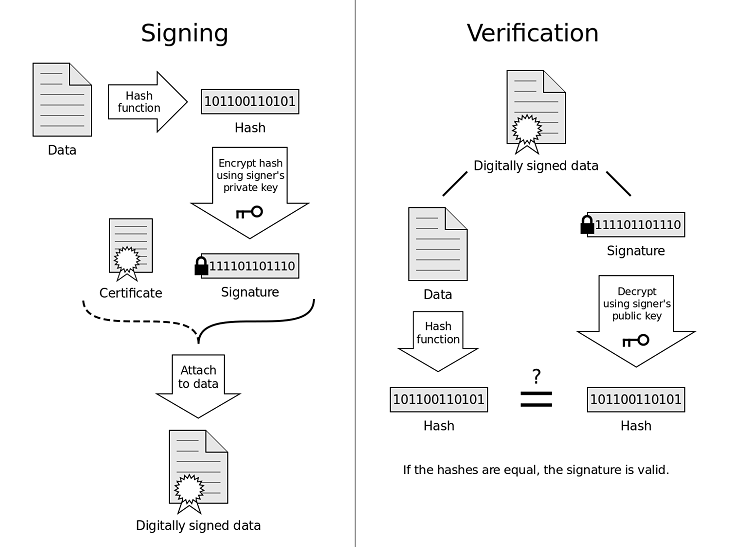


Figure 1 – Explanations of steps for signing and verification

As you can see for signing, first we need to hash our data, for hashing we use SHA-256 function to hash input file from user. Then, we request for a private key to create a signature. Next step is to get a sting array – it’s a binary representation (8-bit) of the hash calculated.

Encryption:

For example we have the following plaint text (or binary representation of it)

\* 01001001 00111100 10111011

And, a private key like the following:

\* {2568, 3845, 3825, 3785, 3705, 969, 3223, 2581}

The encrypted data should be

* 01001001: (0 \* 2568) + (1 \* 3845) + (0 \* 3825) + (0 \* 3785) + (1 \* 3705) + (0 \* 969) + (0 \* 3223) + (1 \* 2581) = 10131
* 00111100: (0 \* 2568) + (0 \* 3845) + (1 \* 3825) + (1 \* 3785) + (1 \* 3705) + (1 \* 969) + (0 \* 3223) + (0 \* 2581) = 12284
* 10001011: (1 \* 2568) + (0 \* 3845) + (0 \* 3825) + (0 \* 3785) + (1 \* 3705) + (0 \* 969) + (1 \* 3223) + (1 \* 2581) = 12077

So, the encrypted data is: 10131 12284 12077

Next, we save encrypted data in bin file in selected destination path . Also, I’ve implemented a check for selected path if it contains already a signature file and ask user to replace it or not.

*#Verification of signature:*

As showing in “Figure 1” the verification process is to hash document with the same hashing algorithm and decrypt signature with public key, and compare the two result if they are matching the signature is verified and data is fine. If not the data has been manipulated and it’s not verified.

The verification process in the program takes 2 parameters first is for path to signature file and second is for document (data). We first request the sender public key for the decryption of signature. Then, we get two last parameters of the public key which are n and m values, and we need to find the n inverse , which is multiplicative inverse of n mod m.

n x n inverse mod (m) = 1

Taking the previous encrypted data as example and a public key like the following:

\* 11-27-57-117-237-478-960-1923 1287-3863

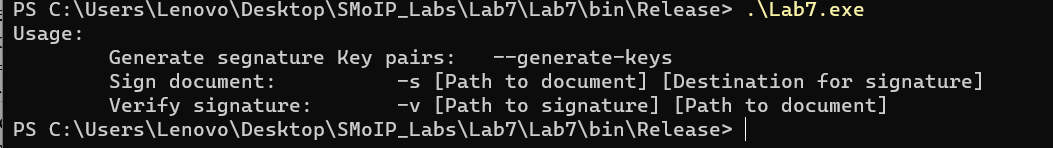
n inverse in this case is: 1930

1930\* 10131 mod (3863) = 2187

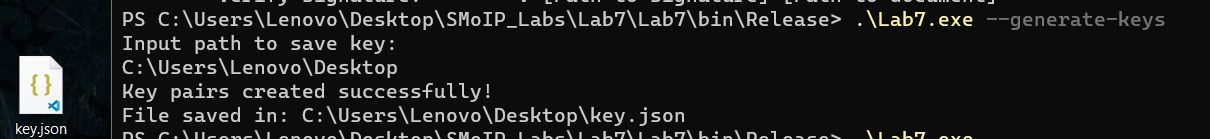
27 + 237 + 1923 = 2187

So, make that corresponding bits 1 and others 0 which is 01001001. And the same way with the rest of data and in this way we can decrypt and get the hash from signature. Next step is to hash the file that we have with the same hash algorithm and compare the two results.

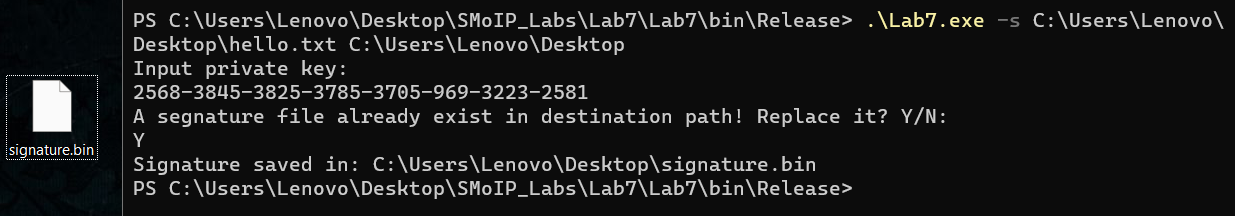
***Screenshots of the program***:



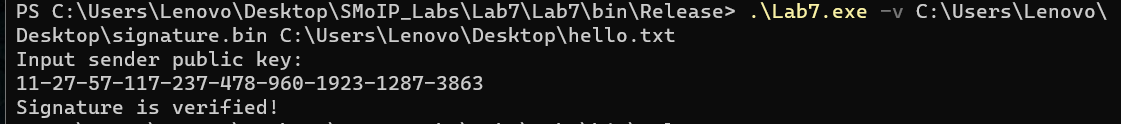
*Figure 2 – Usage of the program*

**

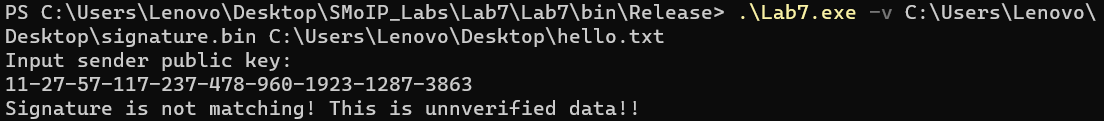
*Figure 3 – Generate key pairs*

**

*Figure 4 – Sign a document*

****

*Figure 5 – verify the signature*

****

*Figure 6– Manipulate the file and verify the signature*

**Source Code:**

*<https://github.com/dikejr99/SMoIP_Labs/tree/main/Lab7>*

**Conclusions:**

For this laboratory work, I have gained principles of developing a program for creating an electronic digital signature with a created Asymmetric encryption algorithm, and verify a signature with original data.