***CSE4057 Information Systems Security – Spring 2022***

***Doc. Dr. Omer Korcak***

***Homework l***

# Generation of public-private key pairs

1. By using the Crypto library in Python, we have created an RSA public-private key pair. It has 2048 bits. We have exported these into .bin files so that we can keep them securely to be used later. You may find the files as such.

Ka+ value shown below in Figure 1. File for reference; **ka\_public.pem**

Text

Description automatically generated

Figure . Public key

Ka- value shown below. File for reference; **ka\_private.pem**

A screenshot of a computer

Description automatically generated with medium confidence

Figure . Private key

1. Elliptic-Curve Diffie Helman public private key-pairs are created using a repository as reference. The repository itself is commented in the code. It gets the curve named *“secp256r1”,* and create secrets by using *secrets* library randbelow function. We get both Kb and Kc values. Once we complete the generation of these keys, we get the public keys of the private keys by multiplying these private keys with curve values. Here you can find the values for Kb and Kc.Text

   Description automatically generated

Figure . ECDH keys

# Generation of symmetric keys

1. We have created one 128-bit and one 256-bit symmetric key by using the os module of Python. It basically randomly creates these keys for us, and we do base64 encoding to get the keys in the format that we can use for encryption. You may see the generated keys below in Figure 4.

Text

Description automatically generated

Figure 4. Generated symmetric keys and encryption & decryption

In addition to creation of these keys, we encrypt these values with Ka that we created in the first question, you may see the output in Figure 4 above.

1. We multiply the public key of Kc and private key of Kb and get K3. We also multiply the public key of Kb and private key of Kc and get K3. We get the same key as it is shown in Figure 4 above.

# Generation and Verification of Digital Signature

You may see our whole work in one screenshot. We have a lorem ipsum dolor text above 1000 characters. We get the hashed string of it thanks to SHA256 algorithm and then encrypt & decrypt the hashed string to show m, H(m) and the digital signature (therefore, verification) on the screen. You may see the details below in Figure 5. You can see that the encrypted and decrypted values are the same. The digital signature is in the file named ***“encrypted\_hash\_string.bin”***.

Text

Description automatically generated

Figure . Output for Generation and Verification of Digital Signature

# AES Encryption

For this question, we will be printing only the logs into output. You see the values on the screen in Figure 6 below. We will elaborate the work over that screenshot. You may see the input file having 1035 KB as size in the file named “fourth\_question\_input.txt”.

You will see the files as follows. There will be one output for each item and one verification (decrypted version of them) file for each item. The files are as follows;

fourth\_question\_input.txt\_i.encrypted & fourth\_question\_verification.txt\_i 🡪 Encrypted & decrypted version of i.  
fourth\_question\_input.txt\_ii.encrypted & fourth\_question\_verification.txt\_ii 🡪 Encrypted & decrypted version of ii.  
fourth\_question\_input.txt\_iii.encrypted & fourth\_question\_verification.txt\_iii 🡪 Encrypted & decrypted version of iii.

fourth\_question\_input.txt\_i\_new.encrypted & fourth\_question\_verification.txt\_i\_new 🡪 Encrypted & decrypted version of i with different IV value.

Text

Description automatically generated with medium confidence

Figure . AES output

## Detailed explanation

1. AES (128 bit key) in CBC Mode
   1. We encrypted the file and the output is in fourth\_question\_input.txt\_i.encrypted.
   2. We have decrypted that and the output is in fourth\_question\_verification.txt\_i.
   3. The time elapsed is 3 milliseconds (22:53:30.077 start, 22:53:30.107 end)
   4. You may see the same encryption with different values in the following files
      1. fourth\_question\_input.txt\_i\_new.encrypted
      2. fourth\_question\_verification.txt\_i\_new
2. AES (256 bit key) in CBC mode
   1. We encrypted the file and the output is in fourth\_question\_input.txt\_ii.encrypted.
   2. We have decrypted that and the output is in fourth\_question\_verification.txt\_ii.
   3. The time elapsed is 2 milliseconds (22:53:30.12 start, 22:53:30.14 end)
3. AES (256 bit key) in CTR mode
   1. We encrypted the file and the output is in fourth\_question\_input.txt\_iii.encrypted.
   2. We have decrypted that and the output is in fourth\_question\_verification.txt\_iii.
   3. The time elapsed is 2 milliseconds (22:53:30.15 start, 22:53:30.17 end)

# Message Authentication Codes

1. Here we defined a message as INFORMATION SYSTEMS SECURITY and are going to use this as the input for the message. We have used k1 symmetric key as key for the HMAC SHA256 generation. You may see the details below in Figure 7. The first value is the HMAC object of the message mentioned above.



Figure 7. HMAC SHA256

1. We have applied the same HMAC SHA256 with k1 to the k2 (key k2). You see the output in the second line in Figure 7 above. It is a new 256 bit key.