T.C.
MARMARA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE4057 Information Systems Security - Homework 1
May 10, 2022

REPORT

Group Members
150119901 - Murat Fidan
150119742 - Süleyman Burak Özcan
150119698 - Hüseyin Hasılcı

Lecturer
Doç.Dr. ÖMER KORÇAK

QUESTION NUMBER	STATUS
Question 1	Done
Question 2	Done
Question 3	Done
Question 4	Done
Question 5	Done

Part 1: Generation of public-private key pairs

1.a) RSA public-private key pairs are generated by Crypto library in Python. $KA+$ (public key) and $KA-$ (private key) are printed to terminal and screenshot which is below is taken. Their length is 1024 bits:

```
(b'-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDCvvFeT'
 b'0kEDYBDB/2DyoKZr6ag\nPE2zsS4ckpxFiHAK4cM5B+dmQudwviXWTXAF6PfYQ44D5csDT3DQ'
 b'MgVvRPdt8VcG\nRyvm+Eg4G4WZU7psfuKyuB6quxLgpnAA/jAnRhPRlcksKObvIvvgalz2jMT'
 b'Aq7FX\n9iD5Sp/ZjB04xRnXLwIDAQAB\n-----END PUBLIC KEY-----')
(b'-----BEGIN RSA PRIVATE KEY-----\nMIICWwIBAAKBgQDCvvFeT0kEDYBDB/2DyoKZr6ag'
 b'PE2zsS4ckpxFiHAK4cM5B+dm\nQudwviXWTXAF6PfYQ44D5csDT3DQMgVvRPdt8VcGRyvm+Eg'
 b'4G4WZU7psfuKyuB6q\nuxLgpnAA/jAnRhPRlcksKObvIvvgalz2jMTAq7FX9iD5Sp/ZjB04xR'
 b'nXLwIDAQAB\nAoGAAmOdSm7mkmon/KqIbal6WLBS9vxGZ8HwsuEJkcDOvxNt9bEnIZYIiNWMW'
 b'Uix\n6yCkib6qa1nSV8QNukAi2O6bqnQaS20ubKR2oq0AZuDgvYbGzLXC6sNKEgohMga/\nyro'
 b'tJDot0AIrDzEFv6QixtrgU6ihMr5QtxPx5qP68aAlBGECQQDFVK3e0XE1kY8P\nnmnRtVYIP/'
 b'iMMScoVBUx2Xo7uISkbSA9vAOBwF0RywWFhyRVDHg86iKStLMPFDab\nIjDUIIgdAkEA/KWFJ'
 b's8NrUqs+6O4pxObEzBDi5GlQj28G61m77O3BQlFTTZ9/g3w\nSjR281lTmgzPIhZfkrg/p34T'
 b'shFx8WfyuwJAVuWfW4vnyqs60Kn1939fT2q8TSAo\nGj5MxxL6HOp4nt/fXtA4yx6m3XsGB3M'
 b'nsLw5BrokV25zm6RPF6nKzt80kQJAXZmU\n8wPStVjtLW1Cg+OnmDxRSevzpc7pWfesIzWepK'
 b'cCndCKbQ0M8PDvAMkfR/tm4eIY\nFmtcadkzwszjweQY8QJATBpYxJMXB8+wle8XHSewQeNie'
 b'MjPYZ0lowRHncFxH7jN\nR5sEoODynvtFHO8mRJwHid/bXuKHeDGtpDgjIPQ6zA==\n-----EN'
 b'D RSA PRIVATE KEY-----')
```

Şekil a. KA + and KA −

1.b) Elliptic-Curve Diffie Helman (ECDH) public-private key pairs are generated ($KB+$, $KB-$) and ($KC+$, $KC-$) Private keys are determined by us and public keys are generated with ECDH algorithm. They are printed to the terminal and the screenshot which is below is taken:

```
Kb+:
61911892679843434144313073973433850928647087588536109873363239750696645061269
Kb-:
61
Kc+:
79590375460912417549106840590293229965457820981213700036097620975604886359943
Kc-:
52
```

Şekil a. (KB +, KB −) and (KC +, KC −)

Part 2: Generation of Symmetric keys

2.a) Two symmetric keys which are K1 and K2 are generated by AES function. K1 is 128 bit and K2 is 256 bit. They are printed to the terminal and the screenshot which is below is taken:

```
********************************** k_1: **********************************
(b'+S82ub7WoGVWQbkVnAP9NeZslRTyS+HoQx6M1xqrrXE89EePU+psKPhEo4NqCb8OouXH5JvBNqwY'
 b'Baif0YY+ZHbjc2EObj2uXlzzyIFYdqctarvEAcnNUv2KDgw4ZGXZFVKwdx6FpJBsV64uSY0j2lBb'
 b'mbg+9niRfVtPJv2DUIw=')
********************************** k_2: **********************************
(b'SEu/XNMe6nZwFeo2k0az1etdRbnuxv3Dl+oXVD9wiBrBkyGEFx/xlR0b7dLASTfA/v2fukBOlnqF'
 b'1vVDomXVbs8o4oKESZOhbK56FZ54xTQSLiubiykE1eTT32CyvPKZ7QJpVYPlVcAP+hODcQn9aQTg'
 b'Cq/JZ7k2YyqsK6h3TzelM7pE+DBg3cqbdJZWPT/JzIZ+ehH1EyyJQx3ySp6vWyt1AtQRp8q6fvAT'
 b'3erGs4V4DhkcMJIEPSIO5e6kLwA/Hm7uqHD95IbPpgerSveXIGRI85TqndbPKJwYhKFdbP0juxuA'
 b'j3S9jj31F6+hW9uyRTzlKKCVxM5ZLRFH0zp3YA==')
```

Şekil a. (K1, K2)

Encrypted K1 with *KA* + (public key) is below:

```
Cipher text: b'2\x85\x88P\x1c\x17i\x03\xcd\x88M\xce\x8e)\x1b\xc7~\xaeQsY\xedF\x07\x8aJ\xbe\x08\x
(b'2\x85\x88P\x1c\x17i\x03\xcd\x88M\xce\x8e)\x1b\xc7~\xaeQsY\xedF\x07'
 b'\x8aJ\xbe\x08\xf9\xca\xbf\xca=*B\x88\x0e\x84\x8f\xe5\xd55\x86\x06'
 b'\xea\xf0F\x90\xa6Q}F\x1eG+\x9d\x97\xe9Wa\xa7\xe7\xe3w\x04\xab\xb8MTV=,'
 b'y\xe7\xecG\x17\x17\xff\xa6\xc0\r\xe7\xec\x93\x99|\t\xecR\xf38/\x7f(['
 b'\xea\x08\xe0\xaa\xeb\x85$\xec\x13\xb8\x91\xd3Cg\xbe\xbf\x9a\xf3\x0f\xa2G\tF!'
 b'\xf6\x87Pr\xef\xe3\xe8\x18\xf6$\x04\xcb\x16\xe9\x81^\xb4\xb0\xf9~V\xfep\x1c'
 b'\xe8\x0e\x9bU\xbd\xed\x92i\xf3\xfb\xbb\x84\xcd\xe8QKy6j\x02\t.\xe08'
 b'\xce\xcc\xa8\x18')
```

Şekil a. encrypted K1 with KA +

Encrypted K2 with *KA* + (public key) is printed the terminal and taken screenshot is below:

```
Cipher text: b'\xf8l \xa6\x99\x94\xb4\xb2]\xcdB\x82\xf1\x91Mm| Qu\xa3\x8e\xfb\x15\x05
(b'\xf8l \xa6\x99\x94\xb4\xb2]\xcdB\x82\xf1\x91Mm| Qu\xa3\x8e\xfb\x15'
 b'\x05\x11\xb7m\x00\x91\xddb\xef\xd5)w0\x1fl\xbc\x02\x80\xf0a\x03Y\xb5\xf0'
 b"=\x92h\xe2\x86\xe5OQ\x19\xaf\xd9aA\x97x\x80\xc1\xb4\xf9\xdd\x99\xbd\xce'"
 b'\x81r\xacD \xc6(\xa9\xe2\xdd\x82\xea\xd18\x82\xd5\xd82\x9b!\x05"~8'
 b'W\x94\xf0\xa3\x87\x1c\xe0\xff\xef\xa9\x14\x1d!\x83r\x83\xe1\xa0\x9e\x95'
 b'\x14\x1f\xe0\x0en\x06)s\x8f\xd4\xefk~\x12T#\xb1#G\xc3\x86\xdc\t\x85'
 b'\x19g\x10\x82\x15\xfc2v\xee\xb86\x0e-\x9dEA\xd1\xcd\x8e\xbc\x8e[\x0f\xe0'
 b's\x93\x9d\x08\x14\xcf"_\xeex\xe5\xc3\xbc\x99\xdf\x15b\xdcC\x0c\x0b \x0c\xfb'
 b'$Tf\xa3\xbf\xa7\xe7h\x19\x1c\xd7\x13~2\xcd\xcaF6ert\x8c\xc4u\xe9kua8*\xc6H'
 b'{\x8d+K\xe9\xe6\x89\xad\x19\x9e\xfc\x91%\x01\x01\xeaa\xef\xb5\xdf\xe36sc'
 b'}\x1f;\xd8\xf4\xbd<\x12\x9d\xca\xfd\x03\x1d\xd5y\xa4L]\x040ae?\x19\xc2dTD'
 b'\xbc\x81\xe3B*:\xbe8)\x96\xc5A\xc0\x8b\x15\xe3\x1fk\\\x1c\xaf]\xaf<,Z\xa3?'
 b'^l2xA\xf3`4\x97\xa7\x10\xe5\x14>\xa1\x98?\x05,D\x83,\xf3\x7f\x04CH\xeb'
 b'\xa6\xcfy\xd5\x89\xbe\xa1\xf4\xde\x1b\x96|\x84\x04\x0c\x8b')
```

Şekil a. encrypted K2 with KA +

Decrypted K1 and K2 with $KA-$ (private key) is printed the terminal and taken screenshot is below:

```
********************** k1 **********************
'8bfiZitGPry4Tr/bPuneo+lch9q+Gl4ozao1v2Kp+p4Zh38CUCgKG0AQTwBFyPu9M1/Zl+TjLMxnqCAgfx7kNq74j5EJsjo1ACSm2906BYgzdoXr+yqDJNLMcBE42Evfsqsw10vq+K
********************** k2 **********************
'A1TJ+BTSKf3Xo4sjNHCj8EHnsKWWP6zHZ7qZcZRy4BotH2Ci6dELIyonk6l/Ejnzi3A3aFfiT6A0bx+o9iudIgxkd7NtCz9S1G5Ql5CXrUHqKZH3x5vy0QMlvs6emcfajTSSs85EGy
```

Şekil a. decrypted K1 and K2 with KA -

2.b) 256 bit symmetric key is generated by Elliptic key Diffie Helman using $KC+$ and $KB-$. This is $K3$. Symmetric key is generated by using $KB+$ and $KC-$ and the generated keys are the same. Values are printed in the terminal and taken screenshot is below:

```
k3 : 43106403132277991908672417735560167038853242241143410049409744700577289830607
Value for checking correctness : 43106403132277991908672417735560167038853242241143410049409744700577289830607
Is symmetric keys are equal?: True
```

Şekil a. K3 ($KC+$ and $KB-$*) and Correctness check value (* $KB+$ and $KC-$ *)*

Part 3: Generation and Verification of Digital Signature

Verifying with RSA: To begin with, we generated a key with RSA. Then we created a public key and a private key through this key. Then we hash the message we want to send using SHA256. Then we created a signature using the public and private keys we have. Using this signature, we verified the public key we have.

```
(b'-----BEGIN RSA PRIVATE KEY-----\nMIICWwIBAAKBgQDZ50sUPKxUP9+BsmyOHjA5H7kt'
 b'b2oxh80NXK5+iIArCLZotLEX\nXYKlW6xBdg0T2iiPUrnZvoIZJweiIFY97KQEql+sVdX1A+L'
 b'Xv2GZ2sFIfV38o/cX\ntUj8BXbvhGiWvbm+ivXFTzpNF+bIP2ZEe9QF4c2wnMs7C3zvEQ8weE'
 b'AoXQIDAQAB\nAoGAInqotKFO7p3Uve7/olVAiClu4bOZeBDm71BVBAyRSz3rrxG4W9weChBBZ'
 b'3JI\ni3WfqV4Lrlqot1YnrQ2018OUCCYhKolb0Xop09ckzw+FxstQHLXsdhKyLsIu8JPt\nGX/'
 b'IscjPHokvr88TAi/MAjD12cZA2ANf8QRjDZwSZx6RqrkCQQDhQQlks3k+YqoJ\nQ1XV3pe8py'
 b'3flTnbP94dlG0a124pQPYo8xc/huIhUSdH6Q3YpJL4XL9W9PRmQ/n7\nhuN+ADPbAkEA96Voq'
 b'yoEVpOIYs9Dd8GaQtbmUTcPZNE9ZZitfmkU6Foh9US1QyRo\nkxPLp5k1VG8fRWh+OlWM5TVa'
 b'a2RggV1mJwJAEE1Lp70RbkFnuunjoWnNo3qZ6E99\nc2+o3I0sZD/pGhU4e3gOW4WggfbEmAC'
 b'23tHyTQUxV9K8iVYsFlcJycmr9wJAIbio\nVAZYqOFWBP5sFXaLZuaUXiK9OeE0Fw1/MmNksW'
 b'70iM1eUVI32y8q4BuAo4quG2lr\nJs5XbS6irVTxvyvUqQJAB6339pzmMcMlKM1QtK7eDVRKO'
 b'LXG+mt2RFAMYBOUypHV\nxHRPULegXLncdF5WdSh0JWq4ZH6hSUBMn07cEy4xNw==\n-----EN'
 b'D RSA PRIVATE KEY-----')
(b'-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDZ50sUP'
 b'kxUP9+BsmyOHjA5H7kt\nb2oxh80NXK5+iIArCLZotLEXXYKlW6xBdg0T2iiPUrnZvoIZJwei'
 b'IFY97KQEql+s\nVdX1A+LXv2GZ2sFIfV38o/cXtUj8BXbvhGiWvbm+ivXFTzpNF+bIP2ZEe9Q'
 b'F4c2w\nnMs7C3zvEQ8weEAoXQIDAQAB\n-----END PUBLIC KEY-----')
```

RsaKey(n=134281764243248008258446820917307552518108471467988532114663054804
062728075983906436504965755275877044049898755315381291901476546785112752
745214827751200156242651332731960933650213249732853638643215572344304107
179095187250007088518067004805570341109972629735183480151576944800422278
50059221384744047411659180 7,e=65537,d=2418780577218277823963508736940683365
8487121865199881313053654606740627507163739803209501819445988197583182855
585365154811738874754692098298018844951751187408809433989711649212546882534
592319956192604200277845961528757740627507249036370210093753858607078185
07929245958581330138921324451829005406659660456113,p=103865363749969587588
4379222057933352240483975007756019717035811379405919935678017901705656489
7124564938887244153094311022235468596984377355803771009488911,q=1292844499
7939683889804922931680451342585771442641702894271277505811093594999482794
0681483084761659739012905951747744214670351903859462375697084911907011 37,u=
440062813774023854959467563893624208325514232421311502163495417383904457
090583525611964014755102548891540690336971254888771860389496397523487647
3947593807)

RsaKey(n=134281764243248008258446820917307552518108471467988532114663054804
062728075983906436504965755275877044049898755315381291901476546785112752
745214827751200156242651332731960933650213249732853638643215572344304107
179095187250007088518067004805570341109972629735183480151576944800422278
50059221384744047411659180 7, e=65537)

```
(b'N\xeckZ\x96\x1br+J@\xdf,\x9e\xb6\x17\xc2|\xba\x97\x11\x133\x97\xff'
 b'\x8c\x88q\xaf\x1d\xb2,\xc3\t\xba\xf7U\xfa\xf2}\xf3\xe2c\xbe\xaa?\xf0\xd5#'
 b'\xf5\xc7\xd2`\x1a\xb3\xf3\xc7N*\xbe\xf9\r\x19\x1a\xfb\xe0V\xf5M\xec:\xd8\x0c'
 b'\xf2\xbd\xe6-\xba\x198?\xc7\xb5\xd2\xd7\x00\x9a\xadQ(\xd3\xcd\x1a\xd0-2\xc1'
 b'\xae\x1f\xb1\xc0U\xc4\x9f_\x99l\xf0\xc7\x9a\xfe\xfc8j\xd5\x90\xcd\xed\xc1N|'
 b'\xdf\xcd\x1c\x9e\xea\xb1-I')
(b'4eec6b5a961b722b4a40df2c9eb617c27cba9711133397ff8c8871af1db22cc309baf755faf2'
 b'7df3e263beaa3ff0d523f5c7d2601ab3f3c74e2abef90d191afbe056f54dec3ad80cf2bde62d'
 b'ba19383fc7b5d2d7009aad5128d3cd1ad02d32c1ae1fb1c055c49f5f996cf0c79afefc386ad5'
 b'90cdedc14e7cdfcd1c9eeab12d49')
The signature is valid.
```

Verifying with ecdsa: To begin with, we created a private key using a different model called ecdsa other than RSA. While creating, we did the hashing using SHA256. Then we created a signature over the private key for the message we want to send. After creating, we verified the public key we have. We checked if it was verified in the output.

```
Private Key: <ecdsa.keys.SigningKey object at 0x00000150C0830FD0>
b'\x10[*[\xb2)D\x1d)\xa6\xeb\xa4Bn/"K\xca\xe4\x0e\xea\xd2U9\xd0\xd0\\\x85\x8f\xa3\x1ax\xa6\x97~\x9bz\x1c\r\xfd\xe61\xe0!\xbf\xf3\x92I'
Public Key: VerifyingKey.from_string(b'\x03\xc5\x12\xf7{c\xaeXZ\xe6\xdb$B\x1b\xc8\x02#t\xe4I\xa6\xa2G\x95\xc6', NIST192p, sha256)
Verified: True
```

Part 4: AES Encryption

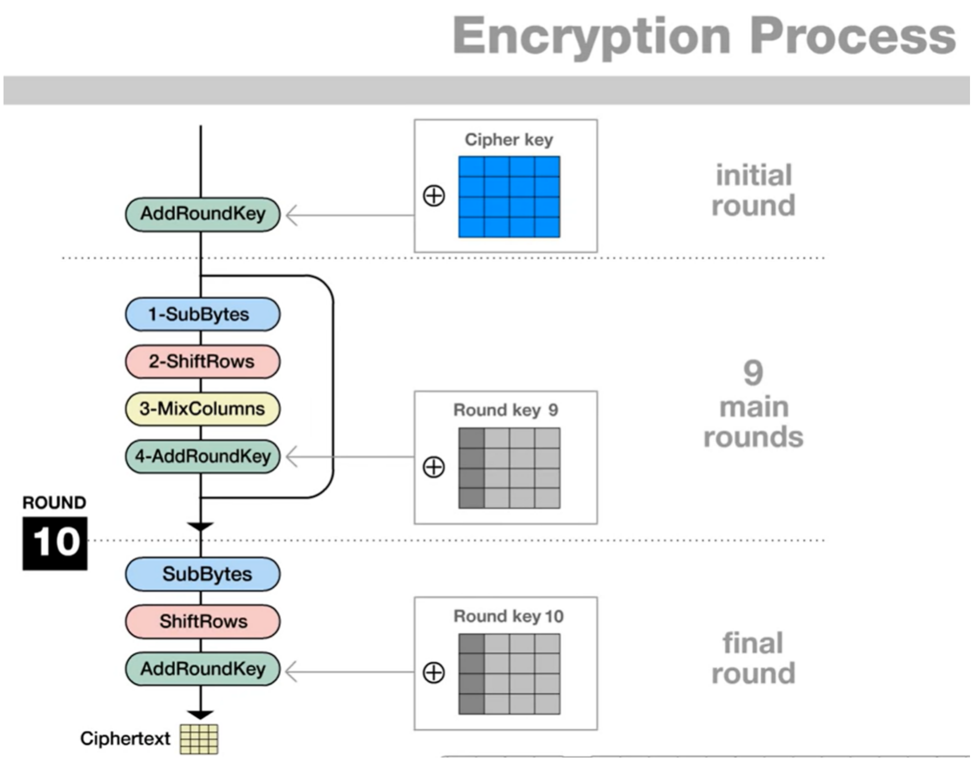
*In this part of the assignment, we will see the steps to encrypt and decrypt a text file using different modes of AES algorithms. For this part, we need a text file on which we will test the AES algorithms. So we downloaded a 1MB size .txt file from the internet. This file is the file named "**sample-text-1mb.txt**" in the folder we uploaded for Homework submission. In order to understand this step well, first of all, it is necessary to understand how the AES algorithm works and its steps. Once we understand the AES algorithm, we will see how it works on different modes. The implementations and outputs of AES algorithms with different modes and different key bits required in the homework are listed in detail below.*

AES Algorithm

There are 4 steps that form the basis of this algorithm:

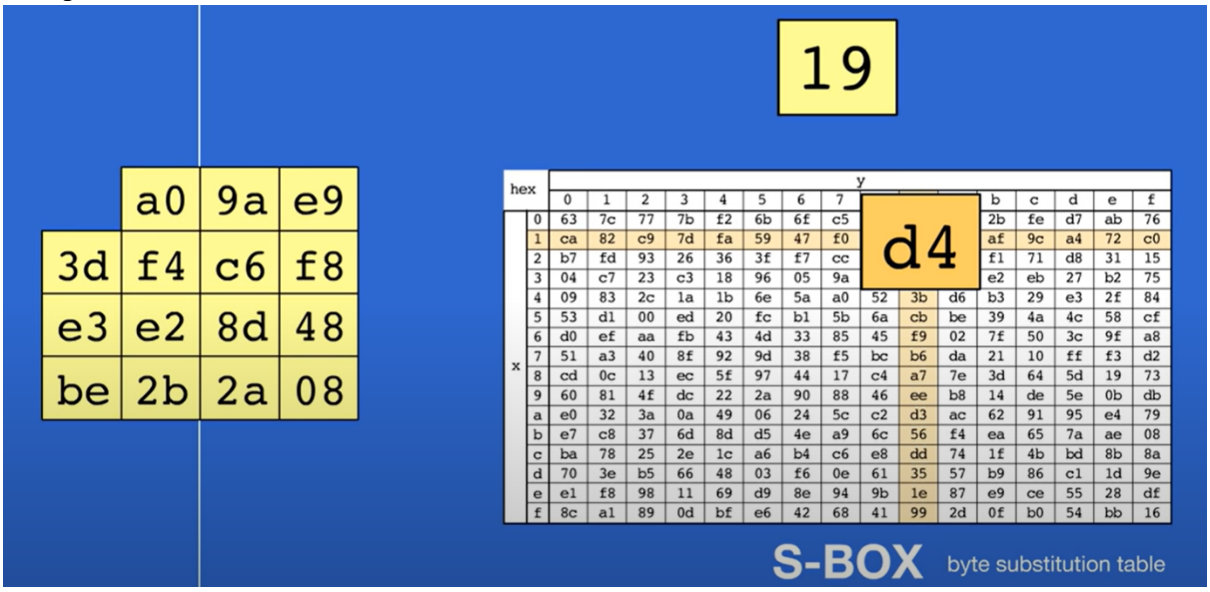
1- SubBytes
2- ShiftRows
3- MixColumns
4-AddRoundKey.

The encryption process in this algorithm is as follows.



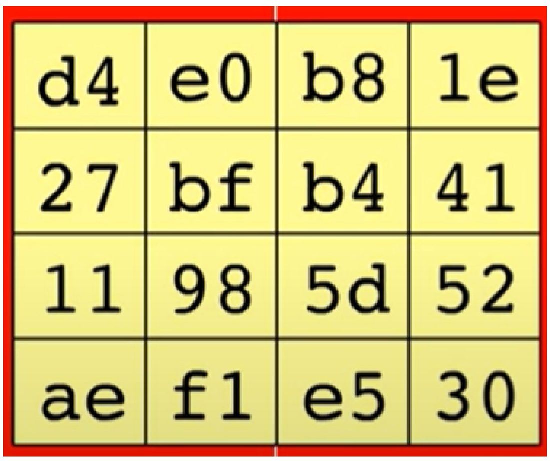
Şekil a. AES Encryption Process Steps

In the SubBytes step, we change the 1-byte parts of our plaintext, which we have made into blocks, with the value it corresponds to in the S-Box. The following example is an example where 1 byte(19 as Hexadecimal) data is exchanged using S-Box.

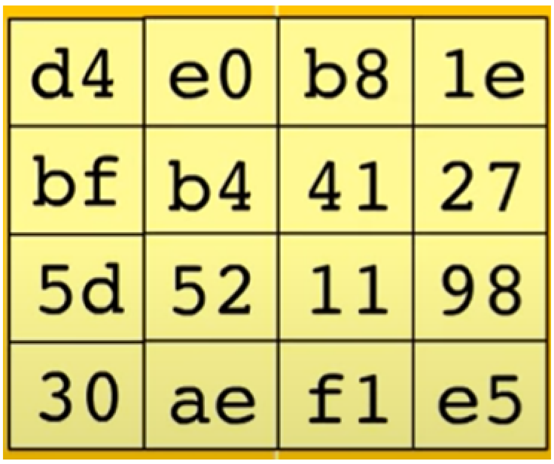


Şekil a. SubBytes Step Example

After all blocks have passed this step, 1 byte of data is changed with the ShiftRows step. We can also call this step the rotation process on the basis of rows. While there is no change in the first line, all bytes are shifted to the left in the second line, all bytes are shifted 2 left in the third line, and all bytes are shifted 3 left in the fourth line. An example is given below. It is given as before and after ShiftRows is applied. We can understand the scrolling operations through these two figures.

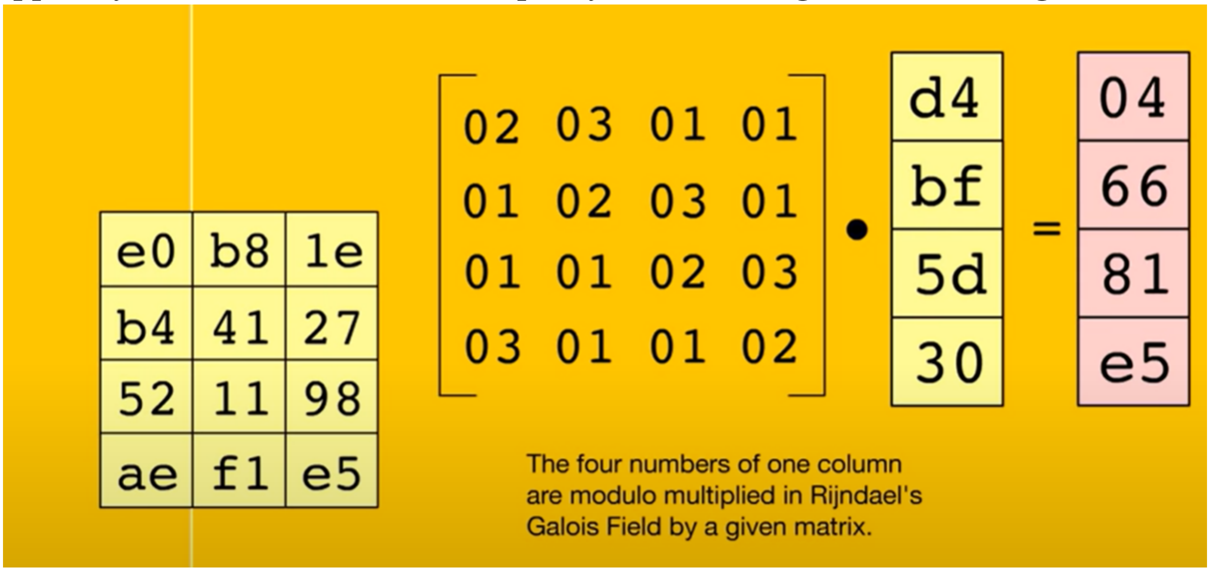


Şekil a. Before Shifting Rows



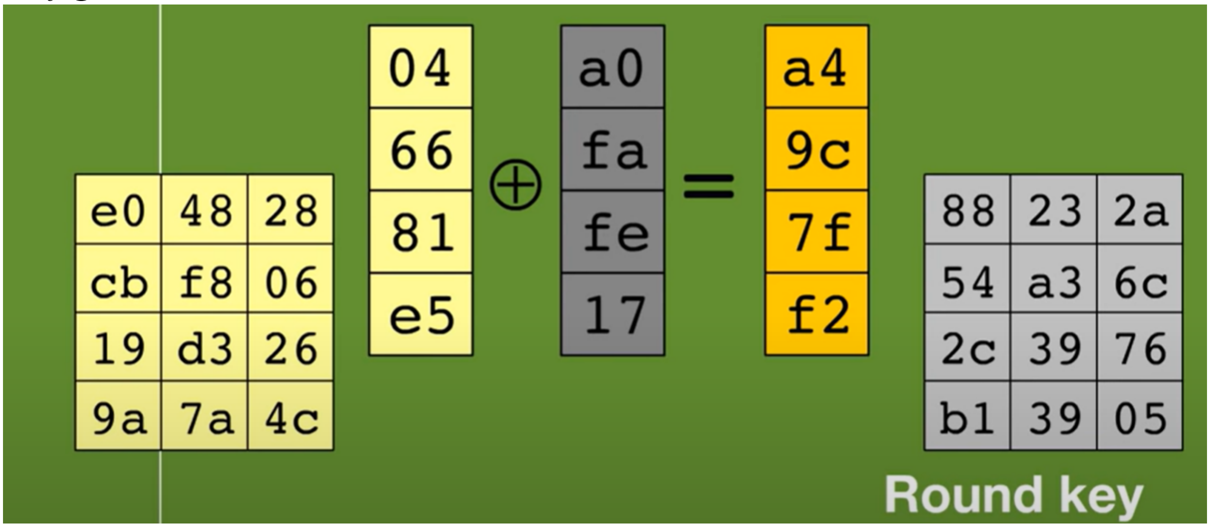
Şekil b. After Shifting Rows

After the rows are shifted, the MixColumns step is performed. In this step, the vector consisting of bytes in the columns is multiplied by the matrix called Rijndael's Galois Field and new column values are obtained. This step is applied for all 4 columns. An example of a column is given in the image below.



Şekil a. MixColumns step example

The key created for that round is added to the byte values obtained as a result of this step. This step is called AddRoundKey. An example for this step is given in the figure below.



Şekil a. AddRoundKey step example

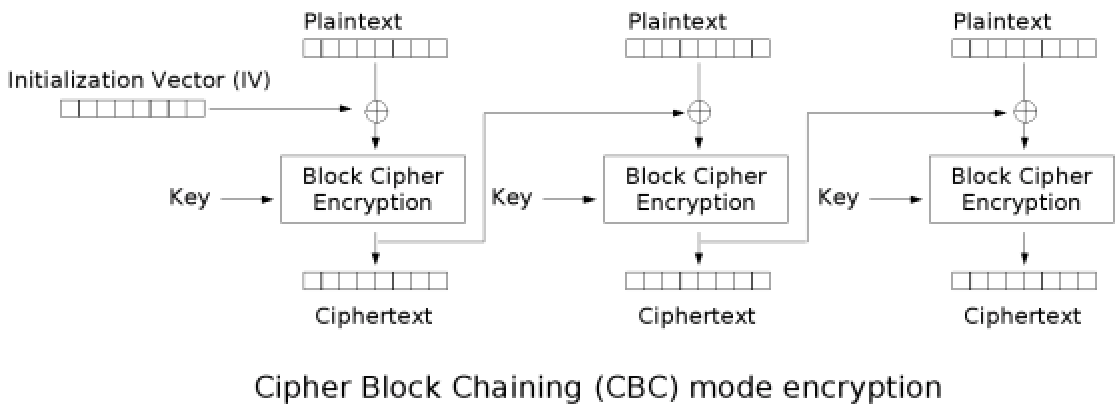
After understanding the steps of the AES algorithm, we can apply this algorithm *according to the desired modes and different key bits in the assignment. During the encryption process, these steps are repeated depending on the key size.*

- *128 bit key - 10 Rounds*
- *192 bit key – 12 Rounds*
- *256 bit key – 14 Rounds*

Since the questions in the homework are over 128 and 256 bit keys, the round numbers will be 10 and 14.

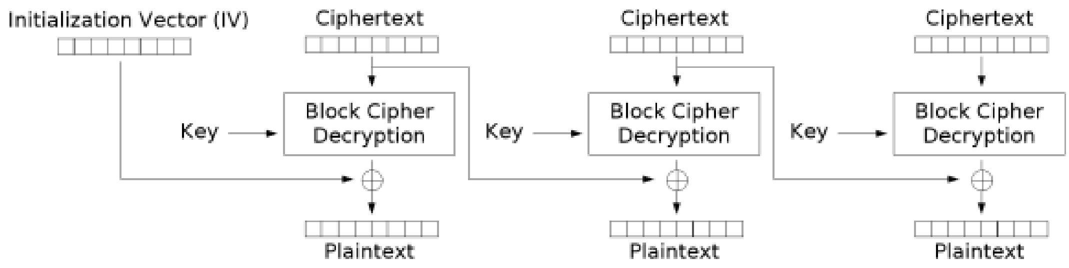
Question 4.1: AES (128 bit key) in CBC mode

We need some inputs to encrypt by applying this algorithm. These are: message to be encrypted, key, initialization vector. These requirements are obtained under the heading "Inputs" in the code section. The initialization vector was created randomly as desired in the homework. After all the requirements were obtained, we started to divide the message to be encrypted into 128-bit packets. We applied the padding for packets with missing bits. After the plaintext was made into 128bit(16 bytes) packages, the AES algorithm was started to be applied with a predetermined key and a randomly generated initialization vector. The visualized version of this process is as follows.



Şekil a. CBC Mode Encryption Diagram

The first block of our plaintext, which we have divided into blocks, enters the XOR process with the randomly determined initialization vector and is processed in the AES algorithm. The ciphertext from this block acts as an initialization vector for the next block. As can be seen in the figure above, in order to encrypt the next block of the message, it is necessary to wait for the ciphertext produced from the previous block. As a result of these steps, our message is encrypted. Although the methods used in the decryption process of the encrypted message are the same, there are some differences. An illustration of the decryption process is given below.



Cipher Block Chaining (CBC) mode decryption

As can be seen in the figure above, the initialization vector used in the encryption process is used in the decryption process. Here, it is ciphertext instead of plaintext that enters the AES algorithm. The result obtained is our original message before encryption. We proved with windows CMD prompt commands that the message obtained as a result of the decryption is the same as the original message. The command "**fc original-text decrypted-text**" prints the differences between the two files. In this process, we have obtained the result that there is no difference between the two files. The output files and screenshots of this question are listed below.

- Plaintext = sample-text-1mb.txt
- Encrypted Text(Ciphertext) = encryption-CBC-128.txt
- Decrypted Text = decryption-CBC-128.txt
- Evidence for no difference between Decrypted text and original = no-difference-CBC-128.png
- Time taken for this process = elapse-time-CBC-128.png

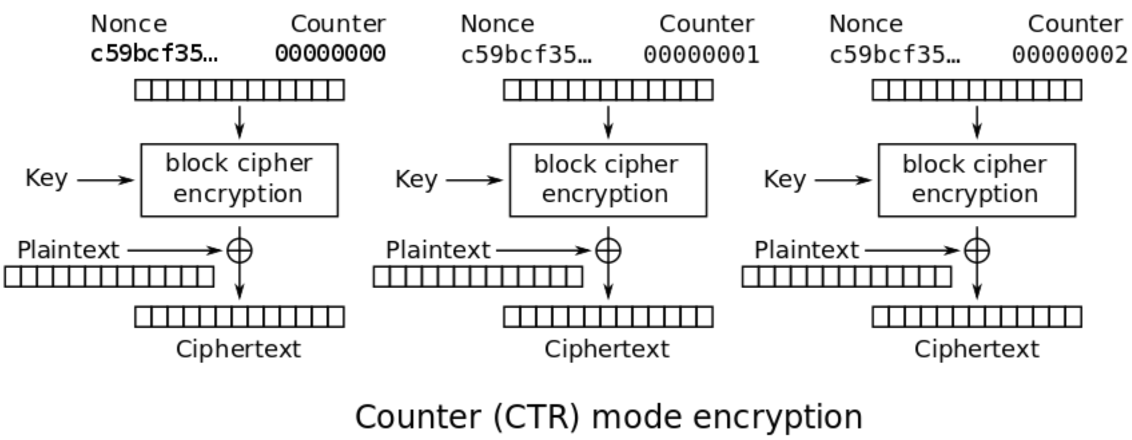
Question 4.2: AES (256 bit key) in CBC mode

Most steps in solving this problem are the same as in "Question 4.1". What is different is the key size and the number of rounds in the AES algorithm that changes depending on this key size. The number of rounds in the AES algorithm is 10 for 128 bits and 14 for 256 bits. The output files and screenshots of this question are listed below.

- Plaintext = sample-text-1mb.txt
- Encrypted Text(Ciphertext) = encryption-CBC-256.txt
- Decrypted Text = decryption-CBC-256.txt
- Evidence for no difference between Decrypted text and original = no-difference-CBC-256.png
- Time taken for this process = elapse-time-CBC-256.png

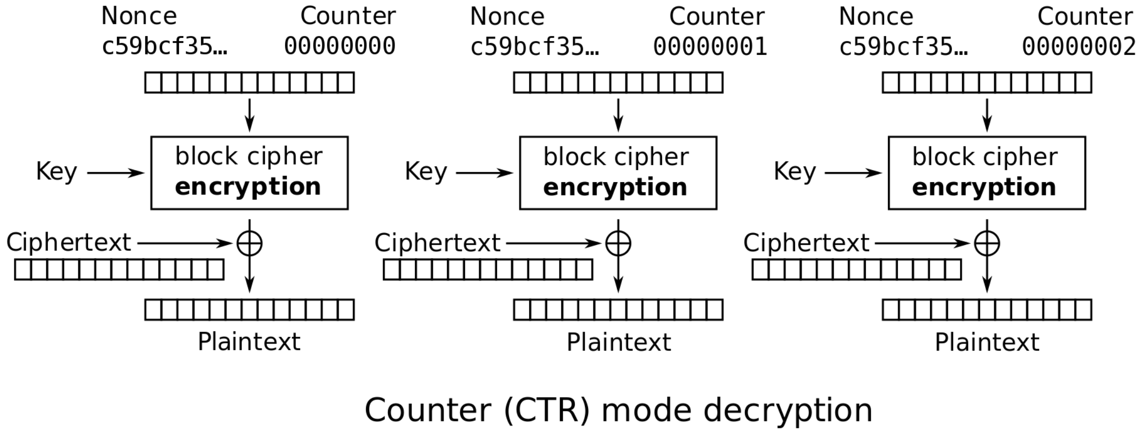
Question 4.3: AES (256 bit key) in CTR mode

The first difference of the solution of this question from the first 2 questions is that instead of the initialization vector, the combination of the nonce and the counter value enters the algorithm in the encryption process. The second difference is that plaintext enters the XOR operation with the encrypted message resulting from the AES algorithm instead of the initialization vector. In addition, the most important difference of the CTR mode from the CBC mode is that the encryption of a block is completely independent of the result of the previous block. In CBC mode, the next block was using the ciphertext resulting from the previous block for encryption. But in CTR mode, encryption of all blocks can be done in parallel. It does not have to wait for the previous block result. A visualization of the implementation of the AES algorithm in CTR mode is given below.



Şekil a. CTR Mode Encryption Diagram

In CTR mode, the decryption process is processed similarly to the steps that occur in the encryption process. The difference is that the result of the AES algorithm is processed XOR with ciphertext instead of plaintext. As a result of these steps, the original message is obtained from the encrypted message. The visualized version of the decryption process is given below.



Şekil a. CTR Mode Decryption Diagram

The output files and screenshots of this question are listed below.

- *Plaintext = sample-text-1mb.txt*
- *Encrypted Text(Ciphertext) = encryption-CTR-256.txt*
- *Decrypted Text = decryption-CTR-256.txt*
- *Evidence for no difference between Decrypted text and original = no-difference-CTR-256.png*
- *Time taken for this process = elapse-time-CTR-256.png*

Question 4.1: AES (128 bit key) in CBC mode after changing IV

In option D of the 4th question, it is requested to change the initialization vector and the changes that occur on the ciphertext as a result of this change. In CBC mode, the encryption steps begin with the initialization vector XORing the plaintext and giving this result to the AES algorithm. Using the result obtained, encryption steps are made for the next block. Because of this dependency, changing the result in one block will change the result in all blocks. In this part of the question we changed the initialization vector and started encryption processes on the same plaintext. The result we obtained was completely different from the ciphertext resulting from the initialization vector we used first. The filenames with the results of messages encrypted using the same plaintext and 2 different initialization vectors are as follows:

- *Encrypted Message(Ciphertext) with IV-1 -> encryption-CBC-128.txt*
- *Encrypted Message(Ciphertext) with IV-2 -> encryption-CBC-128-with-new-IV.txt*

Question 4: Elapsed Time on Different Algorithms and Comments

- ***Mode 1 - AES (128 bit key) in CBC mode***

```
Start Time: 1652057627.5101833 - End Time: 1652057627.5161839
Time Elapsed: 0.006000518798828125
```

Şekil a. Time Elapsed for CBC mode(128 bit key)

- ***Mode 2 - AES (256 bit key) in CBC mode***

```
Start Time: 1652057692.1212626 - End Time: 1652057692.127421
Time Elapsed: 0.006158351898193359
```

Şekil a. Time Elapsed for CBC mode(256 bit key)

- ***Mode 3 - AES (256 bit key) in CTR mode***

```
Start Time: 1652057742.1258307 - End Time: 1652057742.1298304
Time Elapsed: 0.0039997100830078125
```

Şekil a. Time Elapsed for CTR mode(256 bit key)

- ***Comments***
  The times taken to encrypt the message are as given above. As can be seen, the shortest time is when AES (256 bit key) in CTR mode because in CTR mode the message encryption process takes place in parallel for all 128 bit blocks. To encrypt a block in CBC mode, the previous block must be encrypted. The message encrypted in the previous block enters the XOR process in the next block. For this reason, it is expected that the encryption process of the 1st block is finished before the encryption of the 2nd block begins. For the 3rd block, it is expected that the 2nd block encryption is finished. This process continues like this. In CTR mode, the encryption process can be carried out in parallel, since the encryption of blocks is completely independent from each other. CTR mode is the mode in which the encryption process takes the shortest time thanks to parallel processing. The mode with the longest time is the CBC mode of AES (256 bit key). The mode with the longest time is the CBC mode of AES (256 bit key). This is because 256 bits are used for the key instead of 128 bits. SubBytes, ShiftRows, MixColumns and AddRoundKey processes, which are the steps of the AES algorithm, are repeated 10 times for 128 bits and 14 times for 256 bits. Since 14 rounds are required for a 256-bit key, it has the longest duration.

  .

Part 5: Message Authentication Codes

In this part of the assignment we are asked to create a message authentication code. To generate the HMAC-SHA256 code, we need a message, a key and an algorithm as inputs. The list of these requirements is given below.

- *Message = "InformationSystemSecurityHomework1Question5"*
- *Symmetric Key = b'BynrIY7TsAIDYuKlzmZvZkht0Vc9azNO'*
  - *Since this key is randomly generated, a different key will be generated for each run. The result of the current study is given in the report.*
- *Algorithm = Hashlib.sha256*

The message authentication code output obtained with the inputs given above is as follows.

- *Message Authentication Code(HMAC-SHA256):*
  1c842cfa45f1ce4e91db1f874c37eb9d9b9c1dc6f30a39844bc9d12afa06be60

   - *Since the generated HMAC-SHA256 code depends on the inputs, it gives different results with different inputs in different run operations.*

For option B of this question, we are asked to apply HMAC-SHA256 on Key-2. Therefore, the inputs we need for this part of the question are listed below.

- *256 bit Key-2 = (generated for part2 but also we showed how it is created on code for part 5.)*
- *Symmetric Key = We used Key-1 used in option A of this question 5.*
- *Algorithm = Hashlib.sha256*

After the necessary inputs were created, the key-2 was updated as a result of the operation. The following screenshot shows the old and new states for key-2.

```
K2 before applying HMAC-SHA256: 337a7345637a6e35614539536736367670717732354f48724363436132384671
K2  after applying HMAC-SHA256: 1c4d9d8241a0fdf711ec2d17d94c4ab466fe40ab786e0652ccf91127c32078df
```

Şekil a. Key-2 Before and After State Screenshot