

Tema 1

Bită Mihai-Alexandru, 3B3

November 1, 2020

1 Modul de utilizare

Cuvânt înainte: *executarea acestui program necesită instalat un modul de Python[1], iar port-ul 9999 trebuie să fie liber!*

Pentru a utiliza acest program se vor efectua, **în ordine**, următorii pași:

1. Se deschid 3 terminale în directorul programului.
2.
 - (a) În primul terminal (**T1**) se va rula comanda `python server.py`
 - (b) În al doilea terminal (**T2**) se va rula comanda `python client_a.py`
 - (c) În al treilea terminal (**T3**) se va rula comanda `python client_b.py`
3. **T2** va cere, pe parcursul rulării, două input-uri
 - (a) Primul input specifică modul AES dorit de nodul **A**: *CBC* sau *CFB*
 - (b) Al doilea input specifică numele fișierului (cu tot cu extensie) care se dorește a fi criptat de nodul **A**: *nume-fișier.extensie* sau *"locatie-fișier"/nume-fișier.extensie*
4. Sfârșitul execuției este indicat de **T1** printr-un mesaj sugestiv, în timp ce **T3** va afișa conținutul fișierului cerut la pasul anterior. În cazul unei erori, se închid toate cele trei terminale și se reiau pașii în aceeași ordine.

2 Modalitatea de rezolvare

Rezolvarea temei este alcătuită din patru module: `server.py`, `client_a.py`, `client_b.py` și `my_aes.py`.

2.1 Modulul `my_aes.py`

Acest modul implementează cele două modalități de criptare/decriptare ale algoritmului AES precizate în cerință (*CBC* și *CFB*). Modulul utilizează biblioteca PyCryptodome[2] pentru a genera string-uri random de 16 bytes și pentru a asigura funcția de criptare/decriptare. Sunt prezente patru funcții: `cbc_enc()`, `cbc_dec()` - pentru criptarea/decriptarea în mod CBC și `cfb_enc()`, `cfb_dec()` - pentru criptarea/decriptarea în mod CFB.

Toate funcțiile prezentate iau patru parametri:

- *plaintext/ciphertext* - un string de bytes ce urmează a fi criptat sau decriptat
- *key* - cheia ce va fi folosită la funcția de criptare/decriptare
- *iv* - vectorul de inițializare
- *ret_blocks* - True dacă se dorește ca funcția să returneze o listă de blocuri sau False ca funcția să returneze un string de bytes

Mai jos sunt prezentate funcțiile. Explicațiile sunt redactate în comentarii în limba engleză.

```

11 def cbc_enc(plaintext, key, iv, ret_blocks = False):
12     # length of plain text does not divide 16 -> add padding (null bytes at the end)
13     plaintext = pad(plaintext, BLOCK_SIZE)
14     cipher_texts = [iv]
15
16     for i in range(int(len(plaintext)/BLOCK_SIZE)):
17         # processing plain text block by block
18         block = plaintext[i*BLOCK_SIZE : i*BLOCK_SIZE + BLOCK_SIZE]
19
20         cipher = AES.new(key, AES.MODE_CBC, iv)
21
22         # applying encryption function on the result of `current block` XOR `previous ciphertext`
23         ciphertext = cipher.encrypt(byte_xor(block, cipher_texts[-1]))
24
25         cipher_texts.append(ciphertext)
26
27     # remove C_0(iv) from the cipher texts
28     cipher_texts.pop(0)
29
30     if ret_blocks:
31         return cipher_texts
32
33     return b"".join(cipher_texts)

```

Figure 1: funcția de criptare CBC

```

35 def cbc_dec(ciphertext, key, iv):
36     plain_texts = []
37     cipher_texts = [iv]
38
39     for i in range(int(len(ciphertext)/BLOCK_SIZE)):
40         # processing ciphertext block by block
41         block = ciphertext[i*BLOCK_SIZE : i*BLOCK_SIZE + BLOCK_SIZE]
42         cipher = AES.new(key, AES.MODE_CBC, iv)
43
44         # applying decryption function on the current ciphertext block
45         # and xor'ing the decrypted text with the previous cipher
46         plaintext = byte_xor(cipher.decrypt(block), cipher_texts[-1])
47
48         cipher_texts.append(block)
49         plain_texts.append(plaintext)
50
51     return unpad(b"".join(plain_texts), BLOCK_SIZE)

```

Figure 2: funcția de decriptare CBC

```

53 def cfb_enc(plaintext, key, iv, ret_blocks = False):
54     # length of plain text does not divide 16 -> add padding (null bytes at the end)
55     plaintext = pad(plaintext, BLOCK_SIZE)
56     cipher_texts = [iv]
57
58     for i in range(int(len(plaintext)/BLOCK_SIZE)):
59         # processing plain text block by block
60         block = plaintext[i*BLOCK_SIZE : i*BLOCK_SIZE + BLOCK_SIZE]
61
62         cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=8*BLOCK_SIZE)
63
64         # applying encryption function on the previous ciphertext and
65         # xor'ing the encrypted text with the current block of plain text
66         ciphertext = byte_xor(cipher.encrypt(cipher_texts[-1]), block)
67
68         cipher_texts.append(ciphertext)
69
70     # remove C_0(iv) from the cipher texts
71     cipher_texts.pop(0)
72
73     if ret_blocks:
74         return cipher_texts
75
76     return b"".join(cipher_texts)

```

Figure 3: funcția de criptare CFB

```

78 def cfb_dec(ciphertext, key, iv):
79     plain_texts = []
80     cipher_texts = [iv]
81
82     for i in range(int(len(ciphertext)/BLOCK_SIZE)):
83         # processing ciphertext block by block
84         block = ciphertext[i*BLOCK_SIZE : i*BLOCK_SIZE + BLOCK_SIZE]
85         cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=8*BLOCK_SIZE)
86
87         # applying decryption function on the previous ciphertext block
88         # and xor'ing the decrypted text with the current cipher
89         plaintext = byte_xor(cipher.decrypt(cipher_texts[-1]), block)
90
91         cipher_texts.append(block)
92         plain_texts.append(plaintext)
93
94     return unpad(b"".join(plain_texts), BLOCK_SIZE)

```

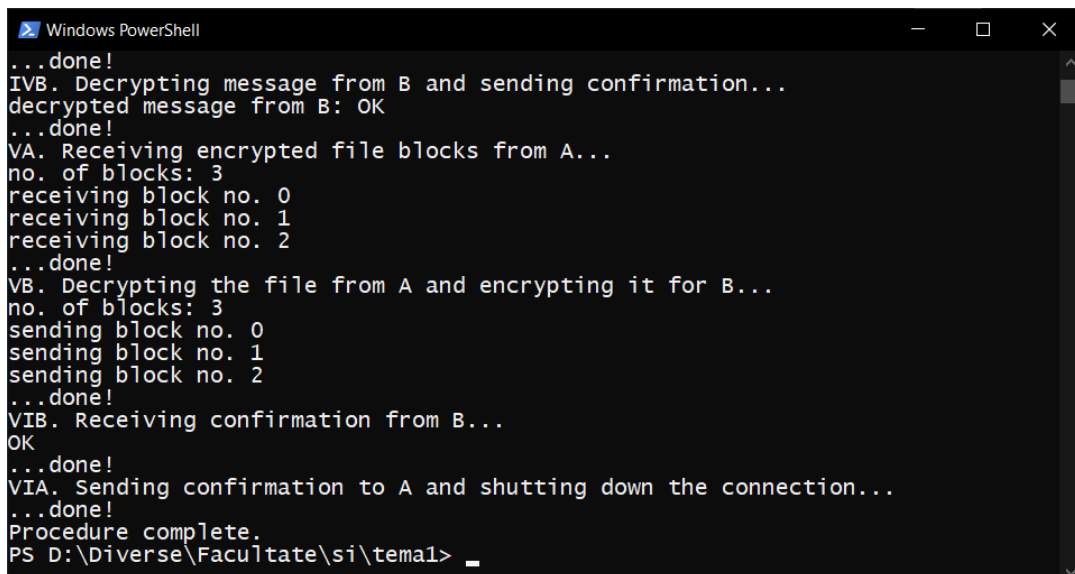
Figure 4: funcția de decriptare CFB

2.2 Arhitectura server-client

Modulul `server.py` creează un server TCP ce servește maximum doi clienți (nodurile **A** și **B**) folosind *thread-uri*. Astfel, putând partaja memoria, se cunoaște etapa la care se află rezolvarea problemei în orice instanță de timp.

Fiecare *thread* se ocupă de comunicarea cu nodul care i-a fost atribuit. Un nod se conectează la server prin modulul `client.a.py` sau `client.b.py`. Primul care se va conecta va fi nodul **A**, așadar o va face prin `client.a.py`, iar **B** prin `client.b.py`.

Conexiunile se închid automat atunci când procedeul s-a sfârșit cu succes sau manual atunci când a fost întâmpinată o eroare. Toate etapele sunt jurnalizate pe ecranul fiecărui terminal.



```

Windows PowerShell
...done!
IVB. Decrypting message from B and sending confirmation...
decrypted message from B: OK
...done!
VA. Receiving encrypted file blocks from A...
no. of blocks: 3
receiving block no. 0
receiving block no. 1
receiving block no. 2
...done!
VB. Decrypting the file from A and encrypting it for B...
no. of blocks: 3
sending block no. 0
sending block no. 1
sending block no. 2
...done!
VIB. Receiving confirmation from B...
OK
...done!
VIA. Sending confirmation to A and shutting down the connection...
...done!
Procedure complete.
PS D:\Diverse\Facultate\si\tema1>

```

Figure 5: rezultatul unei proceduri complete la ecranul lui **T1**

```
Windows PowerShell
received encrypted IV1: b'&\x1a\x0b\xa4\x8c\xa1\x85\x1b0@N\x84x\x147\x80\xe7H ^
\t(J\xfa\xe8$\x9e\x86\xe6\xc8\xd6\n\x12'
decrypted K1: b'\x08\xa5PG\xf0\xc0\x07\xa4\xcf\xfa\xaf(\x9c\xd5s\xf4'
decrypted IV1: b'\x13;\x1d\xfe\xdc\xd3,\x07IC\xfd\x0f\x7fn\x07T'
...done!
IV. Sending encrypted confirmation to KM...
sending message: b'\x06i\xb9LD\xdb\x94_\x92U!\x015\xf0\xa2)'
...done!
V. Receiving confirmation from KM...
OK
...done!
-- COMMENCING PHASE 2/2
I. Encrypting file and sending every block of it to KM...
file to encrypt: secret.txt
no. of blocks to be sent: 3
sending block no. 0
sending block no. 1
sending block no. 2
...done!
II. Receiving confirmation from KM that B decrypted the entire file...
OK
...done!
PS D:\Diverse\Facultate\si\tema1>
```

Figure 6: rezultatul unei proceduri complete la ecranul lui **T2**

```
Windows PowerShell
received encrypted K2: b"'\xe6.\x8e\xe2'\xd4m\x05\xbb\x5\x06\x8by&h5\x16\xc8\x ^
e3\xe4y\xbd\xba\x8c\xc0Z\xdd\x8d6\x16\xc7s"
received encrypted IV2: b';kyy\xd2\x12!\xab\x03\xaa\xbd\xd3Fj\xe6\xf9\xcb\x8d\
x14\x7f\x8c{\xf6"xEf\xd5\tZI\xbf'
decrypted K2: b'G\x9a\xb0\xb0,\xeb\x9a\xad\xe2y\xf9\xa8\x101\xefZ'
decrypted IV2: b'\x9a\xdfg+\xd9-\xd6\x03ZfB\xfo/}a\x96'
IV. Sending encrypted confirmation to KM...
sending message: b'\xf0G\xe2\xa4$\x86[R3\xd1\x90X\x17\xbc\x00j'
...done!
V. Receiving confirmation from KM...
OK
...done!
-- COMMENCING PHASE 2/2
I. Receiving encrypted blocks from KM...
no. of blocks to be received: 3
received block no. 0
received block no. 1
received block no. 2
...done!
II. Decrypting every block and sending confirmation message to KM...
nuclear nukes codes: 123456, qwerty, password
...done!
PS D:\Diverse\Facultate\si\tema1>
```

Figure 7: rezultatul unei proceduri complete la ecranul lui **T3**

References

- [1] Python <https://www.python.org/>
- [2] PyCryptodome <https://pycryptodome.readthedocs.io/>