

LAB 2: Symmetric Key

Dilan Coral¹

School of Mathematical and Computational Science
Universidad Yachay Tech, Urcuquí - Ecuador

March 08, 2024

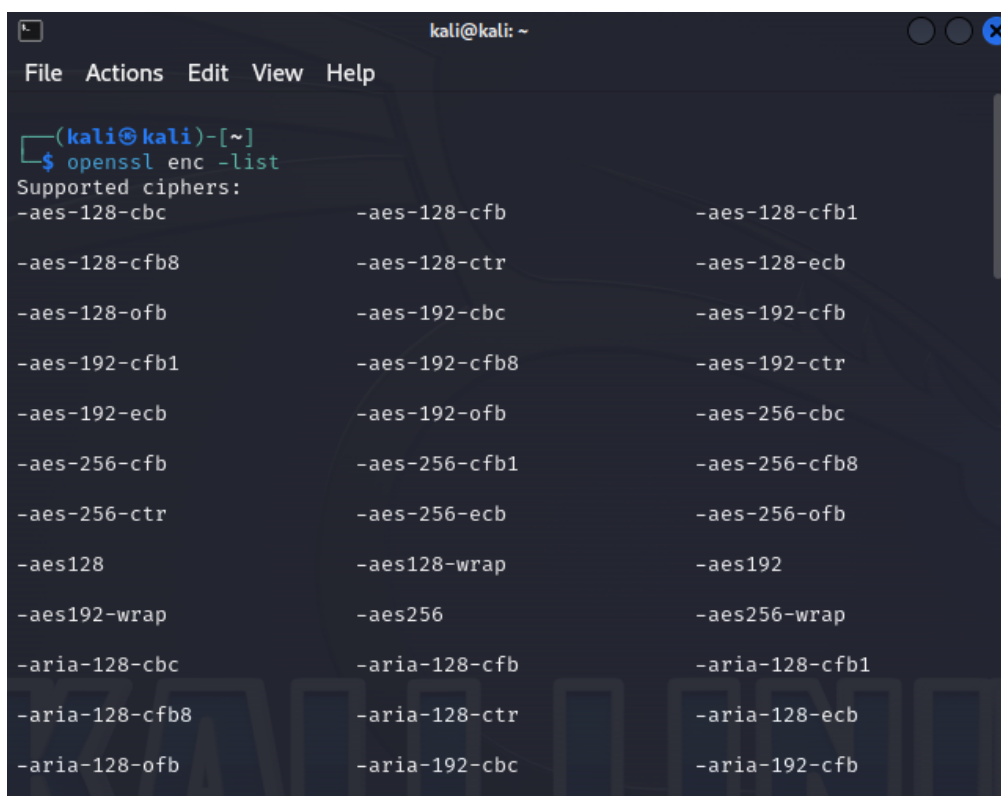
You can find the code for this project on GitHub. Here is the repository link:

- Repository: <https://github.com/Lelis10/Computer-Security---YT.git>

1 Exercise 1: Base-64 encoding, hexadecimal representation, and modulus operator.

1. Outline five encryption methods supported by OpenSSL

```
1 openssl enc -list
```



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ openssl enc -list  
Supported ciphers:  
-aes-128-cbc          -aes-128-cfb          -aes-128-cfb1  
-aes-128-cfb8         -aes-128-ctr          -aes-128-ecb  
-aes-128-ofb          -aes-192-cbc          -aes-192-cfb  
-aes-192-cfb1         -aes-192-cfb8         -aes-192-ctr  
-aes-192-ecb          -aes-192-ofb          -aes-256-cbc  
-aes-256-cfb          -aes-256-cfb1         -aes-256-cfb8  
-aes-256-ctr          -aes-256-ecb          -aes-256-ofb  
-aes128               -aes128-wrap          -aes192  
-aes192-wrap          -aes256               -aes256-wrap  
-aria-128-cbc         -aria-128-cfb         -aria-128-cfb1  
-aria-128-cfb8        -aria-128-ctr          -aria-128-ecb  
-aria-128-ofb         -aria-192-cbc         -aria-192-cfb
```

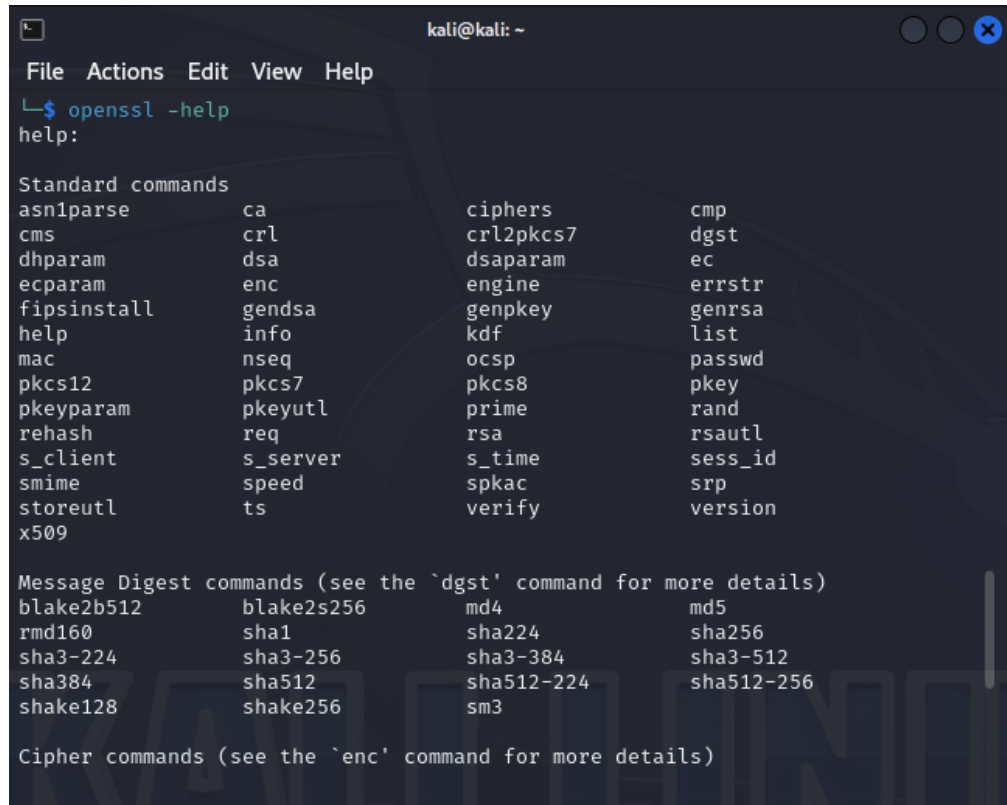
2. Outline the version of OpenSSL

```
1 openssl version
```

```
(kali@kali)-[~]  
$ openssl version  
OpenSSL 3.0.11 19 Sep 2023 (Library: OpenSSL 3.0.11 19 Sep 2023)
```

3. Outline the help of OpenSSL

```
1 openssl -help
```



```
kali@kali: ~  
File Actions Edit View Help  
$ openssl -help  
help:  
  
Standard commands  
asn1parse      ca              ciphers         cmp  
cms            crt            crl2pkcs7       dgst  
dhparam        dsa           dsaparam        ec  
ecparam        enc           engine          errstr  
fipsinstall    gensa         genpkey         genrsa  
help          info          kdf             list  
mac           nseq         ocsf            passwd  
pkcs12         pkcs7        pkcs8           pkey  
pkeyparam      pkeyutl      prime          rand  
rehash        req          rsa             rsautl  
s_client       s_server     s_time         sess_id  
smime         speed        spkac          srp  
storeutl       ts           verify         version  
x509  
  
Message Digest commands (see the `dgst' command for more details)  
blake2b512     blake2s256    md4             md5  
rmd160         sha1          sha224          sha256  
sha3-224       sha3-256      sha3-384        sha3-512  
sha384         sha512        sha512-224     sha512-256  
shake128       shake256       sm3  
  
Cipher commands (see the `enc' command for more details)
```

4. Checking if given numbers are prime

```
1 openssl prime 1111
```

check if the following are prime numbers:

- (a) 16340690919010772729
- (b) 3218553137707046031277850554278875389372
- (c) 108413135436976585570533402883129530221352335126040176869642726536380775978251

```
(kali@kali)-[~]
$ openssl prime 1111
457 (1111) is not prime

(kali@kali)-[~]
$ openssl prime 16340690919010772729
E2C5CBD06C16A6F9 (16340690919010772729) is prime

(kali@kali)-[~]
$ openssl prime 3218553137707046031277850554278875389372
9755EC95B6A8792310A30EEF431225DBC (3218553137707046031277850554278875389372)
is not prime

(kali@kali)-[~]
$ openssl prime 10841313543697658557053340288312953022135233512604017686964
2726536380775978251
EFAFA8FFC80B2385BA78AA8E2FE486C12C74D115BA00D74955546EF81A74D50B (10841313543
6976585570533402883129530221352335126040176869642726536380775978251) is prime
```

5. Generating a random prime number of 4096 bits

```
1 openssl prime -generate -bits 4096
```

```
(kali@kali)-[~]
$ openssl prime -generate -bits 4096
10093346350069668673195795696086023595677666675238140439160406941320732767508
45258629883117243666158100429498362060989695958048749067494225774127094795511
8823846003656346660107724138943423471675944703198071497881472455643041745769
22841705440889805104176515153300714876040064027601549031684401686219288277594
20234759629141332651691772175924391084148786482568719937808706683907448261073
86027147124463254092295810358296529123616607873765184901498328053005032455880
13994469754174150919611487296731945122328671513596524277731859693431914004909
99057911625540714823809116163244717481418212205926533250515117012432693085822
07193339212415989951955094207576522547767725241737731290238075725056547795045
82546572576985076575537654316599030247236307268370936135302117001753061459723
77640079174079688725930490066709487224784814005881190406741532761826069287441
77997908035854222126429482866349945479958328925694192216872572485577056060419
82851877638036589129452871610571656154532596915574653164111229979720557016076
07043427789354364395041236302312730361525427571428233371849101872456928898909
99301749642782562528524956865230324515059564573726916048874863946073684722272
4475018224312312204442000716772879996693288888107050195082662162714234933077
19
```

6. Encrypting a file with aes-256-cbc

```
1 openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin
```

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin

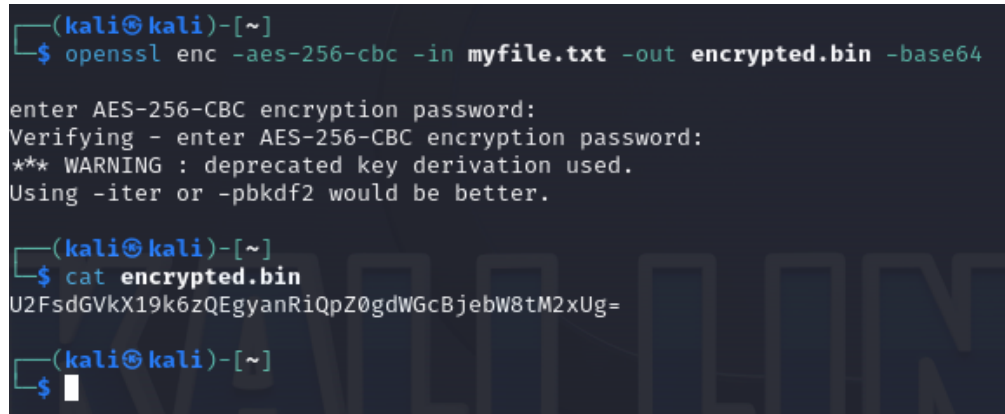
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(kali@kali)-[~]
$ cat encrypted.bin
No%♦♦>

(kali@kali)-[~]
$
```

7. Encrypting a file with aes-256-cbc and base64 encoding

```
1 openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64
```



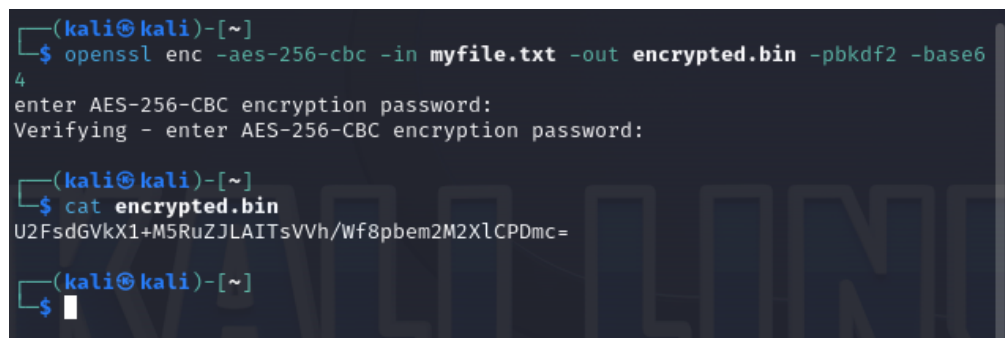
```
(kali㉿kali)-[~]  
$ openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64  
enter AES-256-CBC encryption password:  
Verifying - enter AES-256-CBC encryption password:  
*** WARNING : deprecated key derivation used.  
Using -iter or -pbkdf2 would be better.  
  
(kali㉿kali)-[~]  
$ cat encrypted.bin  
U2FsdGVkX19k6zQEgyanRiQpZ0gdWGcBjebW8tM2xUg=  
  
(kali㉿kali)-[~]  
$
```

8. Encrypting a file with aes-256-cbc, pbkdf2, and base64 encoding

```
1 openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -pbkdf2 -base64
```

(b) Has the output changed? [Yes][No]

(c) Why has it changed?



```
(kali㉿kali)-[~]  
$ openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -pbkdf2 -base64  
enter AES-256-CBC encryption password:  
Verifying - enter AES-256-CBC encryption password:  
  
(kali㉿kali)-[~]  
$ cat encrypted.bin  
U2FsdGVkX1+M5RuZJLAITsVVh/Wf8pbem2M2XlCPDmc=  
  
(kali㉿kali)-[~]  
$
```

(b) Yes, the output has changed.

(c) The output has changed because using PBKDF2 for key derivation introduces additional security measures compared to the default key derivation method. PBKDF2 (Password-Based Key Derivation Function 2) is a key stretching algorithm that iterates a cryptographic hash function multiple times to generate a strong, derived key from a password. This increases the computational complexity, making it harder for attackers to perform brute-force attacks against the encrypted data. Additionally, base64 encoding converts binary data into ASCII text, which can change the representation of the encrypted output.

9. Decrypting the encrypted file

```
1 openssl enc -d -aes-256-cbc -in encrypted.bin -pbkdf2 -base64
```

(b) Has the output been decrypted correctly? Yes, it decrypt correctly.

(c) What happens when you use the wrong password? The command prompt throws an error.

```
(kali㉿kali)-[~]
└─$ openssl enc -d -aes-256-cbc -in encrypted.bin -pbkdf2 -base64
enter AES-256-CBC decryption password:
Computer Security

(kali㉿kali)-[~]
└─$ openssl enc -d -aes-256-cbc -in encrypted.bin -pbkdf2 -base64
enter AES-256-CBC decryption password:
bad decrypt
40D78ED4A47F0000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad
decrypt:../providers/implementations/ciphers/ciphercommon_block.c:129:
VE+/\+,|♦l♦♦`

(kali㉿kali)-[~]
└─$
```

10. Encrypting a file with Blowfish and attempting decryption

```
1 # Encrypt with Blowfish
2 openssl enc -bf -in myfile.txt -out encrypted_blowfish.bin
3
4 # Attempt decryption
5 openssl enc -d -bf -in encrypted_blowfish.bin -out decrypted_blowfish.txt
```

```
(kali㉿kali)-[~]
└─$ openssl enc -bf -in myfile.txt -out encrypted_blowfish.bin
enter BF-CBC encryption password:
Verifying - enter BF-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(kali㉿kali)-[~]
└─$ openssl enc -d -bf -in encrypted_blowfish.bin
enter BF-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Computer Security
```

(a) Did you manage to decrypt the file? Yes

11. Encrypting a file with 3DES and attempting decryption

```
1 # Encrypt with 3DES
2 openssl enc -des3 -in myfile.txt -out encrypted_3des.bin
3
4 # Attempt decryption
5 openssl enc -d -des3 -in encrypted_3des.bin -out decrypted_3des.txt
```

```
(kali㉿kali)-[~]
$ openssl enc -des-ede3 -in myfile.txt -out encrypted_3des.bin

enter DES-EDE3-ECB encryption password:
Verifying - enter DES-EDE3-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(kali㉿kali)-[~]
$ openssl enc -d -des-ede3 -in encrypted_3des.bin

enter DES-EDE3-ECB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Computer Security

(kali㉿kali)-[~]
$
```

(a) Did you manage to decrypt the file? Yes

12. Encrypting a file with RC2 and attempting decryption

```
1 # Encrypt with RC2
2 openssl enc -rc2 -in myfile.txt -out encrypted_rc2.bin
3
4 # Attempt decryption
5 openssl enc -d -rc2 -in encrypted_rc2.bin -out decrypted_rc2.txt
```

```
(kali㉿kali)-[~]
$ openssl enc -rc2 -in myfile.txt -out encrypted_rc2.bin

enter RC2-CBC encryption password:
Verifying - enter RC2-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(kali㉿kali)-[~]
$ openssl enc -d -rc2 -in encrypted_rc2.bin

enter RC2-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Computer Security

(kali㉿kali)-[~]
$
```

(a) Did you manage to decrypt the file? Yes

2 Padding (AES)

2.1 Installation of Cryptographic Libraries for AES-256 Encryption

Install the necessary cryptographic libraries for AES-256 encryption.

2.2 Block Size for AES-256 Encryption

With AES which uses a 256-bit key, what is the normal block size (in bytes).

1. Block size (bytes): 16 bytes
2. Number of hex characters for block size: 32 hex characters (each byte represented by 2 hex characters)

2.3 Implementation of AES-256 Encryption with Different Padding Schemes

Demonstrate the implementation of AES-256 encryption with different padding schemes using Python.

2.4 Encryption and Decryption Tests

Perform encryption and decryption tests using sample data and various padding schemes.

```
Lab2 > exercise_2.py > aes_256_decrypt
4
5 def aes_256_encrypt(data, key, padding_method):
6     cipher = AES.new(key, AES.MODE_CBC)
7     if padding_method == 'zero':
8         padded_data = zero_pad(data.encode(), AES.block_size)
9     else:
10        padded_data = pad(data.encode(), AES.block_size, style=padding_method)
11    ciphertext = cipher.encrypt(padded_data)
12    return ciphertext, cipher.iv
13
14 def aes_256_decrypt(ciphertext, key, iv, padding_method):
15     cipher = AES.new(key, AES.MODE_CBC, iv)
16     decrypted_data = cipher.decrypt(ciphertext)
17     if padding_method == 'zero':
18         unpadded_data = zero_unpad(decrypted_data)
19     else:
20         unpadded_data = unpad(decrypted_data, AES.block_size, style=padding_method)
21     return unpadded_data.decode()
22
```

PROBLEMS OUTPUT TERMINAL PORTS

> TERMINAL

```
Padding method: PKCS7
Encrypted data: b0c51a24ac4f7235f816e8fb688e36a2
Decrypted data: Hello, World!

Padding method: X923
Encrypted data: caa48486eabac9372b9a9e8baa9bdf52
Decrypted data: Hello, World!

Padding method: ISO7816
Encrypted data: 276cf995bc3e84752115fc5b7e628c99
Decrypted data: Hello, World!

Padding method: ZERO
Encrypted data: f396f5e6d5bf2877c83a48d31953cdea
Decrypted data: Hello, World!

PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security>
```


3 Padding (DES)

3.1 With DES which uses a 64-bit key, what is the normal block size (in bytes):

- (a) Block size (bytes): 8 bytes
- (b) Number of hex characters for block size: 16 hex characters (each byte represented by 2 hex characters)

3.2 Demonstrate the implementation of DES encryption with different padding schemes using Python.

3.3 Perform encryption and decryption tests using sample data and various padding schemes.

```
Lab2 > exercise_3.py > des_encrypt
1  from Crypto.Cipher import DES
2  from Crypto.Util.Padding import pad
3  from Crypto.Util.Padding import unpad
4  import binascii
5
6  def des_encrypt(data, key, padding_method):
7      cipher = DES.new(key, DES.MODE_ECB)
8      if padding_method == 'zero':
9          padded_data = zero_pad(data.encode(), DES.block_size)
10     else:
11         padded_data = pad(data.encode(), DES.block_size, style=padding_method)
12     ciphertext = cipher.encrypt(padded_data)
13     return ciphertext
14
15 def des_decrypt(ciphertext, key, padding_method):
16     cipher = DES.new(key, DES.MODE_ECB)
17     decrypted_data = cipher.decrypt(ciphertext)
18     if padding_method == 'zero':
19         unpadding_data = zero_unpad(decrypted_data)
20
PROBLEMS  OUTPUT  TERMINAL  PORTS
>  TERMINAL
ts/YT/Informatic Security/Lab2/exercise_3.py"
Padding method: PKCS7
Encrypted data: 5b22ead96e11ff28ebc50c405e0ae518
Decrypted data: Hello, DES!

Padding method: X923
Encrypted data: 5b22ead96e11ff28a2764f52fae3c67f
Decrypted data: Hello, DES!

Padding method: ISO7816
Encrypted data: 5b22ead96e11ff285fc68e06bc4efc46
Decrypted data: Hello, DES!

Padding method: ZERO
Encrypted data: 5b22ead96e11ff28f5e72c004e8ea62b
Decrypted data: Hello, DES!

PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security>
```



```
# cipher=raw input('Enter cipher:') # password=raw input('Enter password:')
```

```

Lab2 > cipher03.py > ...
1  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
2  from cryptography.hazmat.primitives import padding
3  from cryptography.hazmat.backends import default_backend
4
5  import hashlib
6  import binascii
7
8  def pad(data, size=128):
9      padder = padding.PKCS7(size).padder()
10     padded_data = padder.update(data)
11     padded_data += padder.finalize()
12     return padded_data
13
14 def unpad(data, size=128):
15     padder = padding.PKCS7(size).unpadder()
16     unpadded_data = padder.update(data)
17     unpadded_data += padder.finalize()
18     return unpadded_data
19
PROBLEMS  OUTPUT  TERMINAL  PORTS
>  v TERMINAL
a  ● PS C:\Users\de_di\O& C:/Users/de_di/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/de_d
er03.py"
Enter plaintext: DilanCoral
Enter key: hola
Before padding: DilanCoral
After padding (CMS): b'44696c616e436f72616c0606060606'
Cipher (ECB): b'5710a7d04dea51416c7c7332a3c7c99c'
Decrypted: DilanCoral
○ PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```

5 Python Coding (Decrypting)

- 5.1 Modify your coding for 256-bit AES ECB encryption, so that you can enter the cipher text, and an encryption key, and the code will decrypt to provide the result. You should use CMS for padding. With this, determine the plaintext for the following:

CMS Cipher (256-bit AES ECB)	key	Plain text
b436bd84d16db330359edebf49725c62	hello	germany
4bb2eb68fccd6187ef8738c40de12a6b	ankle	spain
029c4dd71cdae632ec33e2be7674cc14	changeme	england
d8f11e13d25771e83898efdbad0e522c	123456	scotland

Tabla 1: CMS Cipher (256-bit AES ECB) Key and Plain text

```

Enter the cipher text (hexadecimal): b436bd84d16db330359edebf49725c62
Enter the encryption key: hello
Decrypted: germany
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:/
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrpyt01.py"
Enter the cipher text (hexadecimal): 4bb2eb68fccd6187ef8738c40de12a6b
Enter the encryption key: ankle
Decrypted: spain
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:/
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrpyt01.py"
Enter the cipher text (hexadecimal): 029c4dd71cdae632ec33e2be7674cc14
Enter the encryption key: changeme
Decrypted: england
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:/
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrpyt01.py"
Enter the cipher text (hexadecimal): d8f11e13d25771e83898efdbad0e522c
Enter the encryption key: 123456
Decrypted: scotland
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```

5.2 Modify your coding for 64-bit DES ECB encryption, so that you can enter the cipher text, and an encryption key, and the code will decrypt to provide the result. You should use CMS for padding. With this, determine the plaintext for the following:

CMS Cipher (64-bit DES ECB)	key	Plain text
f37ee42f2267458d	hello	Germany
67b7d1162394b868	ankle	France
ac9feb702ba2ecc0	changeme	Norway
de89513fbd17d0dc	123456	England

Tabla 2: CMS Cipher (64-bit DES ECB) Key and Plain text

```

Enter the cipher text (hexadecimal): f37ee42f2267458d
Enter the encryption key: hello
Decrypted: Germany
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> &
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt02.py"
Enter the cipher text (hexadecimal): 67b7d1162394b868
Enter the encryption key: ankle
Decrypted: France
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> &
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt02.py"
Enter the cipher text (hexadecimal): ac9feb702ba2ecc0
Enter the encryption key: changeme
Decrypted: Norway
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> &
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt02.py"
Enter the cipher text (hexadecimal): de89513fbd17d0dc
Enter the encryption key: 123456
Decrypted: England
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```

5.3 Update your program, so that it takes a cipher string in Base-64 and converts it to a hex string and then decrypts it. From this now decrypt the following Base-64 encoded cipher streams (Remember to add import base64).

CMS Cipher (256-bit AES ECB)	key	Plain text
/vA6BD+ZXu8j6KrTHi1Y+w==	hello	italy
nitTRpxMhGlaRkuyXWYxtA==	ankle	sweden
irwjGCAu+mmdNeu6Hq6ciw==	changeme	belgium
5I71KpfT6RdM/xhUJ5IKCQ==	123456	mexico

Tabla 3: CMS Cipher (64-bit DES ECB) Key and Plain text

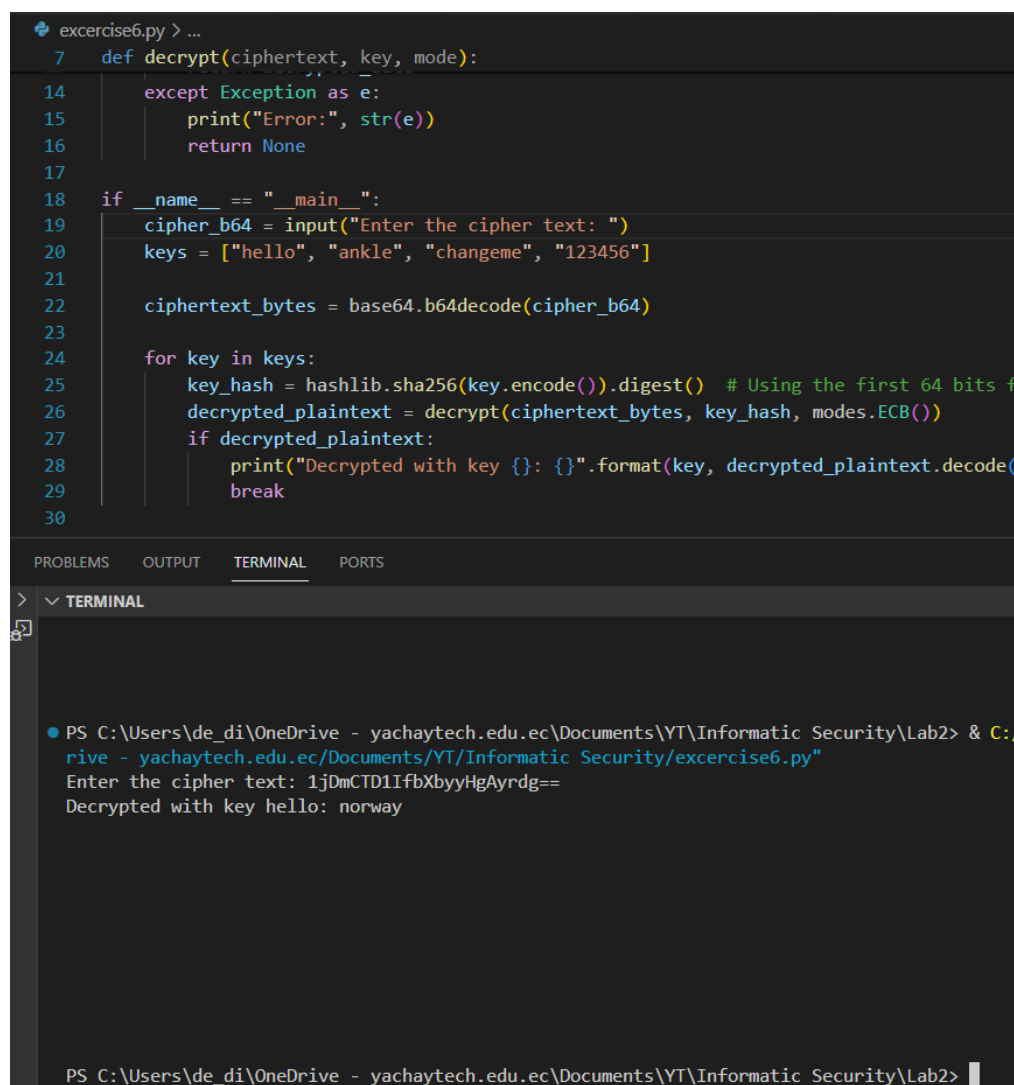
```

Enter the Base64-encoded cipher text: /vA6BD+ZXu8j6KrTHi1Y+w==
Enter the encryption key: hello
Decrypted: italy
● PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt03.py"
Enter the Base64-encoded cipher text: nitTRpxMhGlaRkuyXWYxtA==
Enter the encryption key: ankle
● Decrypted: sweden
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt03.py"
Enter the Base64-encoded cipher text: irwjGCAu+mmdNeu6Hq6ciw==
Enter the encryption key: changeme
Decrypted: belgium
● PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:
rive - yachaytech.edu.ec/Documents/YT/Informatic Security/decrypt03.py"
Enter the Base64-encoded cipher text: 5I71KpfT6RdM/xhUJ5IKCQ==
Enter the encryption key: 123456
Decrypted: mexico
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```

6 Catching exceptions

- 6.1 Implement a Python program which will try various keys for a cipher text input, and show the decrypted text. The keys tried should be: [hello,ankle,changeme,123456]
- 6.2 Run the program and try to crack: 1jDmCTD1IfbXbyyHgAyr dg==
- 6.3 What is the password?



```
exercise6.py > ...
7 def decrypt(ciphertext, key, mode):
14     except Exception as e:
15         print("Error:", str(e))
16         return None
17
18 if __name__ == "__main__":
19     cipher_b64 = input("Enter the cipher text: ")
20     keys = ["hello", "ankle", "changeme", "123456"]
21
22     ciphertext_bytes = base64.b64decode(cipher_b64)
23
24     for key in keys:
25         key_hash = hashlib.sha256(key.encode()).digest() # Using the first 64 bits f
26         decrypted_plaintext = decrypt(ciphertext_bytes, key_hash, modes.ECB())
27         if decrypted_plaintext:
28             print("Decrypted with key {}: {}".format(key, decrypted_plaintext.decode(
29             break
30
```

PROBLEMS OUTPUT TERMINAL PORTS

> ▾ TERMINAL

● PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:/rive - yachaytech.edu.ec/Documents/YT/Informatic Security/exercise6.py
Enter the cipher text: 1jDmCTD1IfbXbyyHgAyr dg==
Decrypted with key hello: norway

PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> |

Figura 2: Enter Caption

7 Stream Ciphers

- 7.1 Develop an application in Python to implement the ChaCha20 stream cipher.
- 7.2 If we use a key of qwerty, nonce = 0x0000000000000000 can you find the well-known fruits (in lower case) of the following ChaCha20 cipher streams:

```

exercise 7.1.py > main
10 def chacha20_decrypt(key, nonce, ciphertext):
11     return plaintext
12
13
14
15 def main():
16     key = b'qwerty' + (32 - len(b'qwerty')) * b'\x00' # Padding the key to 32 bytes
17     nonce = bytes.fromhex('0000000000000000')
18
19     # Ciphertexts provided
20     ciphertexts = [
21         bytes.fromhex('e81461e995'),
22         bytes.fromhex('eb057fe49e34'),
23         bytes.fromhex('e8127ee691315e'),
24         bytes.fromhex('fb0562f592304385d4')
25     ]
26
27     # Decrypt ciphertexts
28     for ciphertext in ciphertexts:
29         plaintext = chacha20_decrypt(key, nonce, ciphertext)
30         print(plaintext.decode('utf-8'))

```

PROBLEMS OUTPUT TERMINAL PORTS

> ▼ TERMINAL

```

PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\exercise 7.1.py
apple
banana
avocado
raspberry
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```

Figura 3: Enter Caption

7.3 RC4 is a standard stream cipher and can be used for light-weight cryptography. Develop an application in Python to implement the RC4 stream cipher.

```

exercise 7.2.py > ...
20
21 def rc4_encrypt(key, plaintext):
22     S = ksa(key)
23     keystream = prga(S, len(plaintext))
24     ciphertext = bytes(bytearray(x ^ y for x, y in zip(plaintext, keystream)))
25     return ciphertext
26
27 def rc4_decrypt(key, ciphertext):
28     return rc4_encrypt(key, ciphertext) # Decryption in RC4 is the same as encryption
29
30 def main():
31     key = b'SecretKey'
32     plaintext = b'Hello, world!'
33
34     # Encryption
35     encrypted_text = rc4_encrypt(key, plaintext)
36     print("Encrypted:", encrypted_text.hex())
37
38     # Decryption
39     decrypted_text = rc4_decrypt(key, encrypted_text)
40     print("Decrypted:", decrypted_text.decode('utf-8'))
41
42 if __name__ == "__main__":
43     main()
44

```

PROBLEMS OUTPUT TERMINAL PORTS

▼ TERMINAL

```

PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2> & C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\exercise 7.2.py
Encrypted: 5cdd502bb05e3ed58b6151228c
Decrypted: Hello, world!
PS C:\Users\de_di\OneDrive - yachaytech.edu.ec\Documents\YT\Informatic Security\Lab2>

```