Experiment 3

Crypto Encryption

Name: Kashish Jain
UID: 2019130022
Class: TE Comps

Objective: -

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

Time Analysis

I used a bat file to use a command to calculate time for each openssl command

timecmd.bat

```
@echo off
@setlocal

set start=%time%

:: Runs your command
cmd /c %*

set end=%time%
set options="tokens=1-4 delims=:.,"
for /f %options% %%a in ("%start%") do set start_h=%%a&set /a start_m=100%%b %% 100&set /a start_s=100%%c %
for /f %options% %%a in ("%end%") do set end_h=%%a&set /a end_m=100%%b %% 100&set /a end_s=100%%c %% 100&se

set /a hours=%end_h%-%start_h%
set /a mins=%end_m%-%start_m%
set /a secs=%end_s%-%start_s%
set /a ms=%end_ms%-%start_ms%
if %ms% lss 0 set /a secs = %secs% - 1 & set /a ms = 100%ms%
if %secs% lss 0 set /a mins = %mins% - 1 & set /a secs = 60%secs%
if %mins% lss 0 set /a hours = %hours% - 1 & set /a mins = 60%mins%
if %hours% lss 0 set /a hours = 24%hours%
if 1%ms% lss 100 set ms=0%ms%

:: Mission accomplished
set /a totalsecs = %hours%*3600 + %mins%*60 + %secs%
echo command took %hours%:%mins%:%secs%.%ms% (%totalsecs%.%ms%s total)
```

Analysis of different Ciphers

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>timecmd openssl enc -aes-128-ecb -e -in task3.txt -out
cipher_aes_ecb.bin -pass pass:hello
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
command took 0:0:0.21 (0.21s total)
```

<u>AES-ECB</u>

Encryption - 0.2s
Decryption – 0.11s

<u>Blowfish</u>

Encryption - 0.07s
Decryption – 0.05s

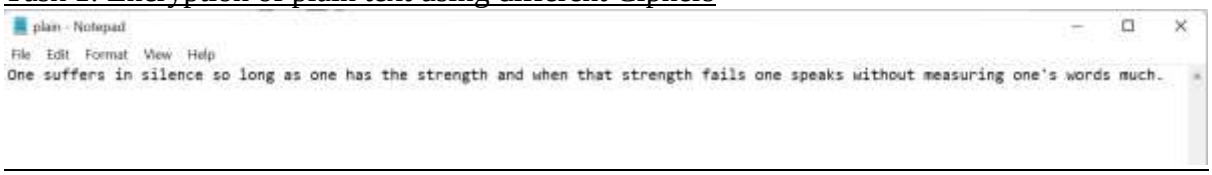
DES
Encryption - 0.06s
Decryption – 0.13s

3DES
Encryption - 0.7s
Decryption – 0.09s

Observations
The results showed that Blowfish has a very good performance compared to other algorithms. Also, it showed that AES has worse performance than 3DES and DES.

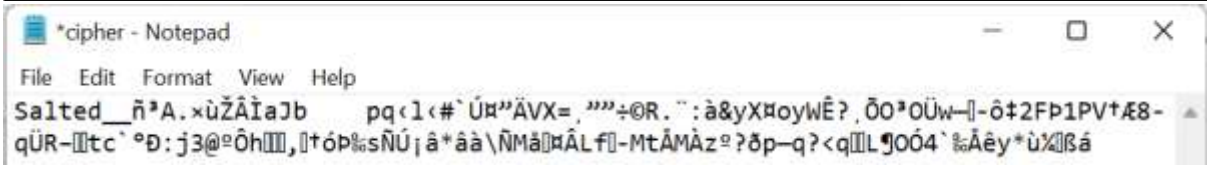
Tasks

Task 1: Encryption of plain text using different Ciphers



1) Encryption Using aes-128-cbc
In this no extra arguments were given.

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128
-cbc -e -in plain.txt -out cipher.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```



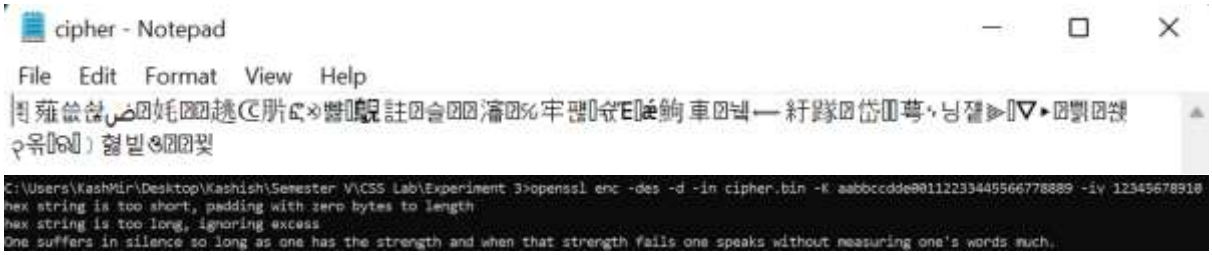
Decryption

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-cbc -d -in cipher.
bin
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
One suffers in silence so long as one has the strength and when that strength fails one speaks without m
easuring one's words much.
```

2) Encryption using des
In this instead of providing a password we provide a KEY and IV. If we do not provide these two arguments, both are generated automatically and can be viewed by adding -p.

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -des -e -in plain.txt -out cipher.bin -K aa
bbccdde00112233445566778889 -iv 12345678910
hex string is too short, padding with zero bytes to length
hex string is too long, ignoring excess
```

Cipher File

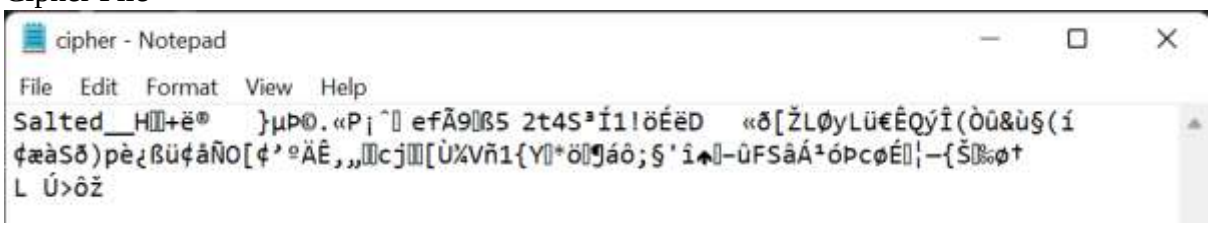


We can observe that here a password is not asked as both KEY and IV is provided

3) Encryption using bf-cbc

Password is provided as argument

Cipher File



Decryption also accepts password as argument

Observation

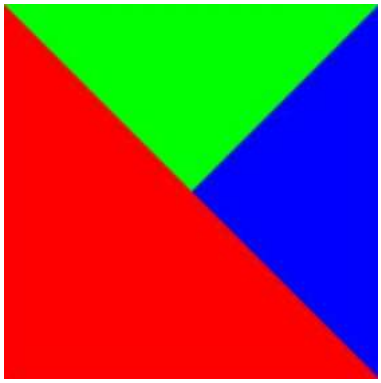
- In AES key length can be 128-bits, 192-bits, and 256-bits while in DES it is 56 bits. Blowfish has a variable key length from 32 bits up to 448 bits
- Blowfish and DES are 64-bit block ciphers, while AES is a 128-bit block cipher; this is a serious issue in a growing number of applications as 64 block ciphers are easier for attackers to crack.

Task 2: Encryption Mode – ECB vs. CBC

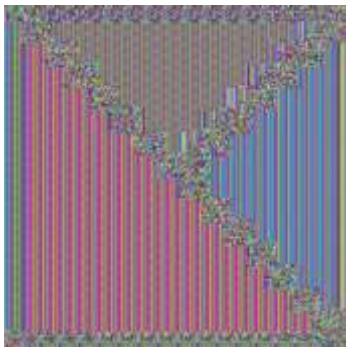
The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

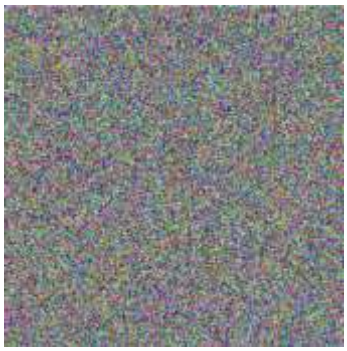
Original Picture



ECB



CBC



```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-256-ecb -e -in in.bmp -out aes_256_ecb.bmp -k 123456 -iv 0102030405060708
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
warning: iv not used by this cipher
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-256-cbc -e -in in.bmp -out aes_256_cbc.bmp -k 123456 -iv 0102030405060708
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>dd if=in.bmp of=aes_256_ecb.bmp bs=54 count=1 conv=notrunc
1+0 records in
1+0 records out
54 bytes copied, 0.0037732 s, 14.3 kB/s

C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>dd if=in.bmp of=aes_256_cbc.bmp bs=54 count=1 conv=notrunc
1+0 records in
1+0 records out
54 bytes copied, 0.0022371 s, 24.1 kB/s
```

Observations:

- I noticed that when using the ECB mode to encrypt the file, the image obtained was not entirely encrypted (the image boundaries remain visible), allowing a third party to readily decipher what the image could be about.

- The CBC mode encryption, on the other hand, completely encrypts the image, making it impossible to see or understand what it was about.
- The fact that the ECB (Electronic Codebook) is essentially the first version of the AES justifies this distinction. It is the most basic type of block cypher encryption, whereas CBC (Cipher Block Chaining) is the most advanced one. With CBC mode encryption, each ciphertext block is dependent on all plaintext blocks processed up to that point. This adds an extra level of complexity to the encrypted data hence harder to decrypt.

Task 3: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this
corruption using ghex.
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. (2) Please explain why. (3) What are the implication of these differences?

ECB

Original Encrypted Text

cipher_aes_ecb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	E6	1B	FA	6E	1C	EB	96	6B	Salted__æ.ún.ë–k
00000010	F9	2B	26	12	56	4E	EC	6F	BA	DF	A3	7E	10	58	35	E8	ù+&.VNìo°ß£~.X5è
00000020	78	4D	3B	C1	29	71	1D	89	09	F7	A2	02	0F	85	67	C0	xM;Á)q.‰.÷¢...…gÀ
00000030	DC	89	F5	CF	9D	DA	D6	C9	9D	10	2B	E5	A1	7A	CE	F7	Ü‰õÏ.ÚÖÉ..+å¡zÎ÷
00000040	FF	18	C8	16	9A	15	64	F2	45	1B	4F	37	2C	CD	20	46	ÿ.È.š.dòE.O7,Í F
00000050	32	6E	39	B2	8E	05	F3	D5	03	06	97	7F	EF	62	F3	E0	2n9²Ž.óÕ..—.ïbóà
00000060	D1	BD	26	3B	01	43	F6	55	19	FB	9C	44	2D	71	D4	5B	Ñ½&;.CöU.ûœD-qÔ[
00000070	C6	76	83	97	D4	4B	44	B7	65	79	20	52	8D	14	1E	CA	Ævƒ—ÔKD·ey R...Ê
00000080	9F	40	66	56	09	63	D3	56	44	7D	78	91	5E	ED	D2	06	Ÿ@fV.cÓVD}x'^íÒ.
00000090	80	0A	89	15	D2	54	E9	CF	4A	58	53	F5	B5	DB	3D	96	€.‰.ÒTéÏJXSõµÛ=–
000000A0	71	B7	30	14	98	09	F8	6F	80	F9	FE	CF	09	B1	1F	34	q·0.˜.øo€ùþÏ.±.4
000000B0	81	5D	EB	BD	E4	C7	F1	C4	F4	EB	66	97	89	93	88	91	.]ë½äÇñÄôëf—‰"ˆ'
000000C0	2D	72	37	40	98	14	42	56	23	38	8A	93	81	D3	55	39	-r7@˜.BV#8Š".ÓU9

Corrupted Encrypted Text

cipher_aes_ecb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	E6	1B	FA	6E	1C	EB	96	6B	Salted__æ.ún.ë–k
00000010	F9	2B	26	12	56	4E	EC	6F	BA	DF	A3	7E	10	57	35	E8	ù+&.VNìo°ß£~.W5è
00000020	78	4D	3B	C1	29	71	1D	89	09	F7	A2	02	0F	85	67	C0	xM;Á)q.‰.÷¢...…gÀ
00000030	DC	89	F5	CF	9D	DA	D6	C9	9D	10	2B	E5	A1	7A	CE	F7	Ü‰õÏ.ÚÖÉ..+å¡zÎ÷
00000040	FF	18	C8	16	9A	15	64	F2	45	1B	4F	37	2C	CD	20	46	ÿ.È.š.dòE.O7,Í F
00000050	32	6E	39	B2	8E	05	F3	D5	03	06	97	7F	EF	62	F3	E0	2n9²Ž.óÕ..—.ïbóà
00000060	D1	BD	26	3B	01	43	F6	55	19	FB	9C	44	2D	71	D4	5B	Ñ½&;.CöU.ûœD-qÔ[
00000070	C6	76	83	97	D4	4B	44	B7	65	79	20	52	8D	14	1E	CA	Ævƒ—ÔKD·ey R...Ê
00000080	9F	40	66	56	09	63	D3	56	44	7D	78	91	5E	ED	D2	06	Ÿ@fV.cÓVD}x'^íÒ.
00000090	80	0A	89	15	D2	54	E9	CF	4A	58	53	F5	B5	DB	3D	96	€.‰.ÒTéÏJXSõµÛ=–
000000A0	71	B7	30	14	98	09	F8	6F	80	F9	FE	CF	09	B1	1F	34	q·0.˜.øo€ùþÏ.±.4
000000B0	81	5D	EB	BD	E4	C7	F1	C4	F4	EB	66	97	89	93	88	91	.]ë½äÇñÄôëf—‰"ˆ'
000000C0	2D	72	37	40	98	14	42	56	23	38	8A	93	81	D3	55	39	-r7@˜.BV#8Š".ÓU9

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-ecb -e -in task3.txt -out cipher_aes_ecb.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-ecb -d -in cipher_aes_ecb.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
┌─┤UIa┐▒g▲¡ W*U▀a stage,
And all the men and women merely players;
They have their exits and their entrances;
And one man in his time plays many parts,
His acts being seven ages.
```

CBC

Original Encrypted Text

cipher_aes_cbc.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	72	37	DC	CF	4D	CF	F1	0A	Salted__r7ÜÏMÏñ.
00000010	C7	25	7A	82	77	6A	DE	5D	A2	1B	92	30	0C	44	DD	55	Ç%z‚wjÞ]¢.’0.DÝU
00000020	61	1A	7E	0A	7E	EF	3E	AC	A2	B6	87	8B	CD	80	55	E1	a.~.~ï>¬¢¶‡‹Í€Uá
00000030	6F	22	21	CF	D3	13	CE	79	3B	F6	C8	92	F4	58	42	9B	o"!ÏÓ.Îy;öÈ’ôXB›
00000040	72	C7	72	7A	B8	D4	5D	02	88	62	D2	F4	7A	D5	E3	DF	rÇrz¸Ô].ˆbÒôzÕãß
00000050	4B	35	DF	96	17	0F	DB	BD	E1	1D	F9	81	5A	74	37	07	K5ß–..Û½á.ù.Zt7.
00000060	ED	8C	C8	80	0F	DD	F0	52	50	9A	F7	9B	69	07	1A	78	íŒÈ€.ÝðRPš÷›i..x
00000070	EE	DE	6D	CA	66	DD	2F	62	BA	B4	9C	DE	49	16	21	D1	îÞmÊfÝ/bº´œÞI.!Ñ
00000080	5B	6B	05	2D	F0	F6	B7	1F	E5	2A	4D	CF	74	3C	AD	3A	[k.-ðö·.å*MÏt<.:
00000090	FF	44	43	29	8D	2C	9D	09	5D	6E	77	F2	C4	21	13	D6	ÿDC).,..]nwòÄ!.Ö
000000A0	10	8E	C8	25	61	0C	33	40	A0	F8	BC	54	69	6A	C7	E5	.ŽÈ%a.3@ ø¼TijÇå
000000B0	C5	AF	50	D8	13	6E	45	AC	6F	41	3B	B2	8D	67	65	C6	Å¯PØ.nE¬oA;².geÆ
000000C0	1C	77	72	14	7C	A9	D7	6C	26	D8	32	0C	56	51	FC	59	.wr.|©×l&Ø2.VQüY

Corrupted Encrypted Text

cipher_aes_cbc.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	72	37	DC	CF	4D	CF	F1	0A	Salted__r7ÜÏMÏñ.
00000010	C7	25	7A	82	77	6A	DE	5D	A2	1B	92	30	0C	44	DD	55	Ç%z‚wjÞ]¢.’0.DÝU
00000020	61	1A	7E	0A	7E	EF	3E	AC	A2	B6	87	8B	CD	80	55	E1	a.~.~ï>¬¢¶‡‹Í€Uá
00000030	6F	22	21	CF	D3	13	CE	79	3B	F6	C8	92	F4	58	42	9B	o"!ÏÓ.Îy;öÈ’ôXB›
00000040	72	C7	72	7A	B8	D4	5D	02	88	62	D2	F4	7A	D5	E3	DF	rÇrz¸Ô].ˆbÒôzÕãß
00000050	4B	35	DF	96	17	0F	DB	BD	E1	1D	F9	81	5A	74	37	07	K5ß–..Û½á.ù.Zt7.
00000060	ED	8C	C8	80	0F	DD	F0	52	50	9A	F7	9B	69	07	1A	78	íŒÈ€.ÝðRPš÷›i..x
00000070	EE	DE	6D	CA	66	DD	2F	62	BA	B4	9C	DE	49	16	21	D1	îÞmÊfÝ/bº´œÞI.!Ñ
00000080	5B	6B	05	2D	F0	F6	B7	1F	E5	2A	4D	CF	74	3C	AD	3A	[k.-ðö·.å*MÏt<.:
00000090	FF	44	43	29	8D	2C	9D	09	5D	6E	77	F2	C4	21	13	D6	ÿDC).,..]nwòÄ!.Ö
000000A0	10	8E	C8	25	61	0C	33	40	A0	F8	BC	54	69	6A	C7	E5	.ŽÈ%a.3@ ø¼TijÇå
000000B0	C5	AF	50	D8	13	6E	45	AC	6F	41	3B	B2	8D	67	65	C6	Å¯PØ.nE¬oA;².geÆ
000000C0	1C	77	72	14	7C	A9	D7	6C	26	D8	32	0C	56	51	FC	59	.wr.|©×l&Ø2.VQüY

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-cbc -e -in task3.txt -out cipher_aes_cbc.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-cbc -d -in cipher_aes_cbc.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
As you like it
And all the men and women merely players;
They have their exits and their entrances;
And one man in his time plays many parts,
His acts being seven ages.
```

CFB

Original Encrypted Text

cipher_aes_cfb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	40	CA	A7	E1	83	B3	8B	1C	Salted__@Ê§áƒ³‹.
00000010	F9	E4	3F	41	DC	BE	AB	F6	AD	A7	22	9D	AA	77	8C	B4	ùä?AÜ¾«ö.§".ª wŒ´
00000020	D1	73	B6	15	8D	75	C8	C6	00	38	89	37	CF	A7	E8	65	Ñs¶..uÈÆ.8‰7Ï§èe
00000030	83	50	E4	1B	92	50	E0	3B	8B	F2	E8	C2	20	BE	6C	FC	ƒPä.'Pà;‹òèÂ ¾lü
00000040	A7	D8	DB	76	B6	E9	BC	E0	7D	91	0B	30	96	60	43	32	§ØÛv¶é¼à}'.0–`C2
00000050	06	61	B0	A3	93	AA	C9	4E	B4	4E	F5	5B	53	21	5E	C8	.a°£"ªÉN´Nõ[S!^È
00000060	F6	B5	E2	F0	17	E2	1E	79	85	C3	D8	6F	4E	C4	5D	53	öµâð.â.y…ÃØoNÄ]S
00000070	7A	C7	60	CB	02	E1	64	D6	DB	B9	CB	BE	F0	F9	EA	3C	zÇ`Ë.ádÖÛ¹Ë¾ðùê<
00000080	C2	F0	18	DA	2D	96	6C	6F	B6	6D	18	E1	B4	EC	C2	C0	Âð.Ú-–lo¶m.á´ìÂÀ
00000090	17	F4	4A	12	DC	D3	E9	20	E8	B1	B2	BC	AC	86	B8	CA	.ôJ.ÜÓé è±²¼¬†¸Ê
000000A0	DF	8B	F2	6E	3D	F0	4B	C2	A2	83	D1	2B	9B	92	34	43	ß‹òn=ðKÂ¢ƒÑ+›'4C
000000B0	9C	FF	EE	00	79	03	3E	56	C0	08	8D	C9	1B	07	95	DB	œÿî.y.>VÀ..É..•Û
000000C0	10	DE	36	28	76	93											.Þ6(v"

Corrupted Encrypted Text

cipher_aes_cfb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	40	CA	A7	E1	83	B3	8B	1C	Salted__@Ê§áƒ³‹.
00000010	F9	E4	3F	41	DC	BE	AB	F6	AD	A7	22	9D	AA	78	8C	B4	ùä?AÜ¾«ö.§".ªx´
00000020	D1	73	B6	15	8D	75	C8	C6	00	38	89	37	CF	A7	E8	65	Ñs¶..uÈÆ.8‰7Ï§èe
00000030	83	50	E4	1B	92	50	E0	3B	8B	F2	E8	C2	20	BE	6C	FC	ƒPä.'Pà;‹òèÂ ¾lü
00000040	A7	D8	DB	76	B6	E9	BC	E0	7D	91	0B	30	96	60	43	32	§ØÛv¶é¼à}'.0–`C2
00000050	06	61	B0	A3	93	AA	C9	4E	B4	4E	F5	5B	53	21	5E	C8	.a°£"ªÉN´Nõ[S!^È
00000060	F6	B5	E2	F0	17	E2	1E	79	85	C3	D8	6F	4E	C4	5D	53	öµâð.â.y…ÃØoNÄ]S
00000070	7A	C7	60	CB	02	E1	64	D6	DB	B9	CB	BE	F0	F9	EA	3C	zÇ`Ë.ádÖÛ¹Ë¾ðùê<
00000080	C2	F0	18	DA	2D	96	6C	6F	B6	6D	18	E1	B4	EC	C2	C0	Âð.Ú-–lo¶m.á´ìÂÀ
00000090	17	F4	4A	12	DC	D3	E9	20	E8	B1	B2	BC	AC	86	B8	CA	.ôJ.ÜÓé è±²¼¬†¸Ê
000000A0	DF	8B	F2	6E	3D	F0	4B	C2	A2	83	D1	2B	9B	92	34	43	ß‹òn=ðKÂ¢ƒÑ+›'4C
000000B0	9C	FF	EE	00	79	03	3E	56	C0	08	8D	C9	1B	07	95	DB	œÿî.y.>VÀ..É..•Û
000000C0	10	DE	36	28	76	93											.Þ6(v"

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-cfb -e -in task3.txt -out cipher_aes_cfb.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-cfb -d -in cipher_aes_cfb.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
All the world(s Eg♦rt▯▯▯▯▯ sn▯▯l the men and women merely players;
They have their exits and their entrances;
And one man in his time plays many parts,
His acts being seven ages.
```

OFB

Original Encrypted Text

cipher_aes_ofb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	95	87	0D	46	00	7D	EE	86	Salted__•‡.F.}î†
00000010	41	2C	DA	90	F2	4D	A1	64	5D	6D	F2	41	DB	AA	9C	3C	A,Ú.òM¡d]mòAÛª œ<
00000020	BA	91	98	E5	E2	4E	01	95	EB	A6	29	75	5D	48	43	02	°'˜åâN.•ë¦)u]HC.
00000030	92	5A	AD	45	45	A5	1B	18	77	D6	4A	40	A1	13	E9	2A	'Z.EE¥..wÖJ@¡.é*
00000040	87	5E	25	7E	0C	F7	DD	6D	74	07	BE	1F	F7	AB	E1	55	‡^%~.÷Ýmt.¾.÷«áU
00000050	33	9C	C4	65	7B	7A	AD	63	1F	D2	40	A1	BB	86	53	82	3œÄe{z.c.Ò@¡»†S,
00000060	B6	A3	76	72	29	62	56	C4	84	13	45	31	70	15	F3	0F	¶£vr)bVÄ„.E1p.ó.
00000070	02	DE	47	0C	40	F6	47	89	11	35	E8	22	AE	F9	11	3D	.ÞG.@öG‰.5è"®ù.=
00000080	DF	37	F0	1F	29	F2	C7	E4	89	A8	EC	71	AB	2A	F3	6B	ß7ð.)òÇä‰¨ìq«*ók
00000090	3D	04	2D	E0	38	79	D1	7D	78	F9	5B	9D	DE	C6	48	74	=.-à8yÑ}xù[.ÞÆHt
000000A0	CF	C3	F9	27	BC	AA	C9	C4	BD	0C	AF	AD	17	E1	F6	1C	ÏÃù'¼ªÉÄ½.¯..áö.
000000B0	70	25	61	2C	5F	33	25	E7	79	1F	A3	5C	32	BE	B8	9F	p%a,_3%çy.£\2¾¸Ÿ
000000C0	AB	6F	C4	A5	11	62											«oÄ¥.b

Corrupted Encrypted Text

cipher_aes_ofb.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	53	61	6C	74	65	64	5F	5F	95	87	0D	46	00	7D	EE	86	Salted__•‡.F.}î†
00000010	41	2C	DA	90	F2	4D	A1	64	5D	6D	F2	41	DB	AB	BC	3C	A,Ú.òM¡d]mòAÛ«¼<
00000020	BA	91	98	E5	E2	4E	01	95	EB	A6	29	75	5D	48	43	02	°'˜åâN.•ë¦)u]HC.
00000030	92	5A	AD	45	45	A5	1B	18	77	D6	4A	40	A1	13	E9	2A	'Z.EE¥..wÖJ@¡.é*
00000040	87	5E	25	7E	0C	F7	DD	6D	74	07	BE	1F	F7	AB	E1	55	‡^%~.÷Ýmt.¾.÷«áU
00000050	33	9C	C4	65	7B	7A	AD	63	1F	D2	40	A1	BB	86	53	82	3œÄe{z.c.Ò@¡»†S,
00000060	B6	A3	76	72	29	62	56	C4	84	13	45	31	70	15	F3	0F	¶£vr)bVÄ„.E1p.ó.
00000070	02	DE	47	0C	40	F6	47	89	11	35	E8	22	AE	F9	11	3D	.ÞG.@öG‰.5è"®ù.=
00000080	DF	37	F0	1F	29	F2	C7	E4	89	A8	EC	71	AB	2A	F3	6B	ß7ð.)òÇä‰¨ìq«*ók
00000090	3D	04	2D	E0	38	79	D1	7D	78	F9	5B	9D	DE	C6	48	74	=.-à8yÑ}xù[.ÞÆHt
000000A0	CF	C3	F9	27	BC	AA	C9	C4	BD	0C	AF	AD	17	E1	F6	1C	ÏÃù'¼ªÉÄ½.¯..áö.
000000B0	70	25	61	2C	5F	33	25	E7	79	1F	A3	5C	32	BE	B8	9F	p%a,_3%çy.£\2¾¸Ÿ
000000C0	AB	6F	C4	A5	11	62											«oÄ¥.b

```
C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 3>openssl enc -aes-128-ofb -d -in cipher_aes_ofb.bin -pass pass:abcd
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
All the world&s a stage,
And all the men and women merely players;
They have their exits and their entrances;
And one man in his time plays many parts,
His acts being seven ages.
```

Observation

- ECB - Since each plaintext block is encrypted and decrypted separately with ecb mode encryption, only the block containing the faulty byte is corrupted; the rest of the text is unaffected. Because there are no dependencies on other blocks in this mode, encryption and decryption can be done by multiple threads at the same time.

- CBC - As we know, in the cbc mode, input and output are chained, thus the plain text block is XORed with the encrypted block from the previous pass, and the chain continues. So I deduced and understood that if one bit of the actual plain block is corrupted, the entire chain will have corrupted bits, resulting in a completely corrupted text, but if only one bit of the ciphertext is corrupted, only two received plaintext blocks will be corrupted, allowing the original data to be recovered.
- CFB - The cfb mode is similar to the cbc mode, with the exception that the previous round's ciphertext must be encrypted before being added to the plaintext bits. Both encryption and decoding must be done with the same encryption technique. I discovered that when one ciphertext bit is corrupted, only the next two plaintext blocks are affected.
- OFB - The keystream bits formed in the case of ofb mode are utilised for the encryption of following data blocks, and as a result, the operation of this mode is identical to that of a conventional stream cypher. In this situation, I discovered that if one bit of a plaintext or ciphertext message is corrupted, only one ciphertext or plaintext bit is also corrupted.

Task 4: Padding

For block ciphers, when the size of the plain text is not the multiple of the block size, padding may be required.

In this task, we will study the padding schemes. Please do the following exercises:

(a) The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.

(b) Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

Original file (20 octets)

task4-20.txt task4-32.txt

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	68	65	72	65	20	61	72	65	20	79	6F	75	20	6E	6F	Where are you no
00000010	3F	3F	3F													w???

Original file (32 octets)

task4-32.txt

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	48	6F	70	65	20	79	6F	75	20	61	72	65	20	69	6E	20	Hope you are in
00000010	70	69	6E	6B	20	6F	66	20	68	65	61	6C	74	68	21	21	pink of health!!

1) CBC

Aes-128-cbc for 20 octets

```
OpenSSL> aes-128-cbc -e -in task4-20.txt -out enc-task4-20.txt -K 00112233445566778
899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cbc -d -in enc-task4-20.txt -out dec-pad-task4-20.txt -K 001122334
45566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cbc -d -in enc-task4-20.txt -out dec-nopad-task4-20.txt -K 0011223
3445566778899aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
```

Encrypted File

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   9D DF 0E 6C E6 4C B5 DD F2 67 60 D2 D1 C0 4B  .ß.læLµÝòg`ÒÑÀK
00000010   CB 31 9B DB A5 B7 05 75 5D F0 D8 0D 85 45 43  Ë1›Û¥·.u]ðØ.…EC
```

Decrypted file (Without padding)

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   68 65 72 65 20 61 72 65 20 79 6F 75 20 6E 6F  here are you no
00000010   3F 3F 3F                                     w???
```

Decrypted file (With padding)

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   68 65 72 65 20 61 72 65 20 79 6F 75 20 6E 6F  here are you no
00000010   3F 3F 3F 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C  w???............
```

Aes-128-cbc for 32 octets

```
OpenSSL> aes-128-cbc -e -in task4-32.txt -out enc-task4-32.txt -K 00112233445566778
899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cbc -d -in enc-task4-32.txt -out dec-pad-task4-32.txt -K 001122334
45566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cbc -d -in enc-task4-32.txt -out dec-nopad-task4-32.txt -K 0011223
3445566778899aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
```

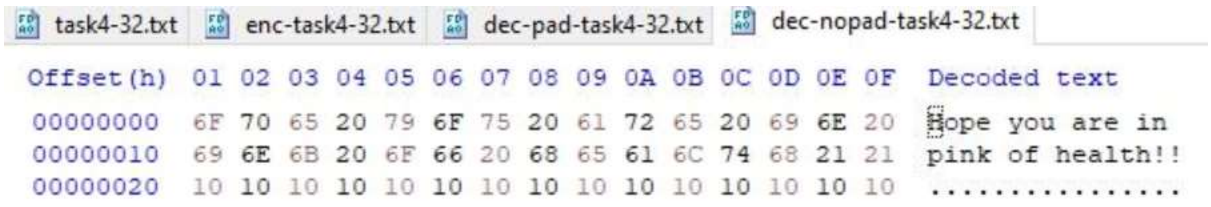
Encrypted file

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   8A D1 B1 F2 55 6A D2 15 12 20 7B 5C F2 9B CD  ÉŠÑ±òUjÒ.. {\ò›Í
00000010   F7 7B 89 7D 44 42 37 76 C7 A9 5F 73 B3 57 C8  ë÷{‰}DB7vÇ©_s³WÈ
00000020   1E A2 02 F7 2F 0F 13 0D 66 C6 83 A0 EC 82 FA  '.¢.÷/...fÆƒ ì‚ú
```

Decrypted file (Without padding)

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   6F 70 65 20 79 6F 75 20 61 72 65 20 69 6E 20  hope you are in
00000010   69 6E 6B 20 6F 66 20 68 65 61 6C 74 68 21 21  pink of health!!
00000020   10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  ................
```

Decrypted file (With padding)



2) ECB

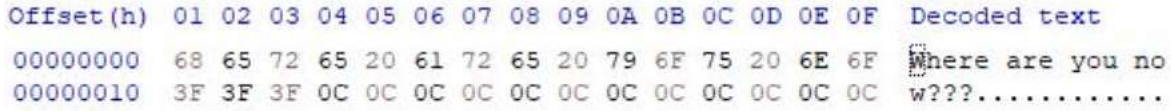
Aes-128-ecb for 20 octets

```
OpenSSL> aes-128-ecb -e -in task4-20.txt -out enc-task4-20-ecb.txt -K 0011223344556
6778899aabbccddeeff
OpenSSL> aes-128-ecb -d -in enc-task4-20-ecb.txt -out dec-pad-task4-20-ecb.txt -K 0
01122334455667788 99aabbccddeeff
OpenSSL> aes-128-ecb -d -in enc-task4-20-ecb.txt -out dec-nopad-task4-20-ecb.txt -K
 00112233445566778899aabbccddeeff -nopad
```

Encrypted File

Decrypted file (Without padding)

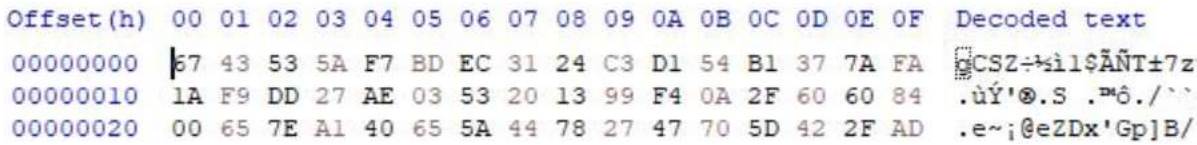
Decrypted file (With padding)



Aes-128-ecb for 32 octets

```
OpenSSL> aes-128-ecb -e -in task4-32.txt -out enc-task4-32-ecb.txt -K 0011223344556
6778899aabbccddeeff
OpenSSL> aes-128-ecb -d -in enc-task4-32-ecb.txt -out dec-pad-task4-32-ecb.txt -K 0
01122334455667788 99aabbccddeeff
OpenSSL> aes-128-ecb -d -in enc-task4-32-ecb.txt -out dec-nopad-task4-32-ecb.txt -K
 00112233445566778899aabbccddeeff -nopad
```

Encrypted file



Decrypted file (Without padding)

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   48 6F 70 65 20 79 6F 75 20 61 72 65 20 69 6E 20  Hope you are in
00000010   70 69 6E 6B 20 6F 66 20 68 65 61 6C 74 68 21 21  pink of health!
```

Decrypted file (With padding)

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   6F 70 65 20 79 6F 75 20 61 72 65 20 69 6E 20  Hope you are in
00000010   69 6E 6B 20 6F 66 20 68 65 61 6C 74 68 21 21  pink of health!!
00000020   10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  ................
```

3) CFB

<u>Aes-128-cfb for 20 octets</u>

```
OpenSSL> aes-128-cfb -e -in task4-20.txt -out enc-task4-20-cfb.txt -K 0011223344556
6778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cfb -d -in enc-task4-20-cfb.txt -out dec-pad-task4-20-cfb.txt -K 0
011223344556778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

Encrypted File

enc-task4-20-cfb.txt dec-pad-task4-20-cfb.txt

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   13 E3 4F 65 E6 6C 26 00 17 80 3C 43 1B 2D B4  .ãOeæl&..€<C.-´
00000010   59 5B B4                                      tY[´
```

Decrypted file (Without padding)

enc-task4-20-cfb.txt dec-pad-task4-20-cfb.txt

```
Offset(h)  01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   68 65 72 65 20 61 72 65 20 79 6F 75 20 6E 6F  here are you no
00000010   3F 3F 3F                                      w???
```

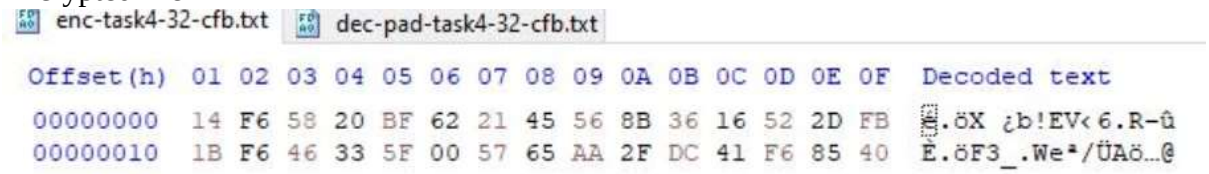
Decrypted file (With padding)

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	68	65	72	65	20	61	72	65	20	79	6F	75	20	6E	6F	here are you no
00000010	3F	3F	3F													w???

Aes-128-cfb for 32 octets

```
OpenSSL> aes-128-cfb -e -in task4-32.txt -out enc-task4-32-cfb.txt -K 0011223344556
6778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-cfb -d -in enc-task4-32-cfb.txt -out dec-pad-task4-32-cfb.txt -K 0
011223344556778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

Encrypted file



Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	14	F6	58	20	BF	62	21	45	56	8B	36	16	52	2D	FB	.öX ¿b!EV‹6.R-û
00000010	1B	F6	46	33	5F	00	57	65	AA	2F	DC	41	F6	85	40	.öF3_.We ª/ÜAö…@

Decrypted file (Without padding)

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6F	70	65	20	79	6F	75	20	61	72	65	20	69	6E	20	ope you are in
00000010	69	6E	6B	20	6F	66	20	68	65	61	6C	74	68	21	21	pink of health!!

Decrypted file (With padding)

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6F	70	65	20	79	6F	75	20	61	72	65	20	69	6E	20	ope you are in
00000010	69	6E	6B	20	6F	66	20	68	65	61	6C	74	68	21	21	pink of health!!

4) OFB

Aes-128-ofb for 20 octets

```
OpenSSL> aes-128-ofb -e -in task4-20.txt -out enc-task4-20-ofb.txt -K 001122334455
6778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-ofb -d -in enc-task4-20-ofb.txt -out dec-pad-task4-20-ofb.txt -K
011223344556778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

Encrypted File

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	13	E3	4F	65	E6	6C	26	00	17	80	3C	43	1B	2D	B4	.ãOeæl&..€<C.-´
00000010	02	C6	09													..Æ.

Decrypted file (Without padding)

enc-task4-20-ofb.txt dec-pad-task4-20-ofb.txt

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	68	65	72	65	20	61	72	65	20	79	6F	75	20	6E	6F	where are you no
00000010	3F	3F	3F													w???

Decrypted file (With padding)

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	68	65	72	65	20	61	72	65	20	79	6F	75	20	6E	6F	where are you no
00000010	3F	3F	3F													w???

Aes-128-ofb for 32 octets

```
OpenSSL> aes-128-ofb -e -in task4-32.txt -out enc-task4-32-ofb.txt -K 0011223344556
6778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL> aes-128-ofb -d -in enc-task4-32-ofb.txt -out dec-pad-task4-32-ofb.txt -K 0
01122334455667788 99aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
OpenSSL>
```

Encrypted file

enc-task4-32-ofb.txt dec-pad-task4-32-ofb.txt

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	14	F6	58	20	BF	62	21	45	56	8B	36	16	52	2D	FB	.öX ¿b!EV‹6.R-û
00000010	54	97	5D	C7	CE	0A	D7	8C	3A	F6	E9	54	94	E8	84	.T—]ÇÎ.×Œ:öéT"è„

Decrypted file (Without padding)

enc-task4-32-ofb.txt dec-pad-task4-32-ofb.txt

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6F	70	65	20	79	6F	75	20	61	72	65	20	69	6E	20	Hope you are in
00000010	69	6E	6B	20	6F	66	20	68	65	61	6C	74	68	21	21	pink of health!!

Decrypted file (With padding)

Offset(h)	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6F	70	65	20	79	6F	75	20	61	72	65	20	69	6E	20	Hope you are in
00000010	69	6E	6B	20	6F	66	20	68	65	61	6C	74	68	21	21	pink of health!!

The size of CBC and ECB encrypted files with the nopad option is 12 bytes more for the 20 bytes file and 16 bytes larger for the 32 bytes file, however the size of OFB and CFB decrypted files is the same, as shown in the screenshots above.

Observations

- I noticed that the size of the ecb and cbc files after decryption increases by 12 bytes for the task4-20.txt, which was originally 20 bytes, and by 16 bytes for the task4-32.txt, which was originally 32 bytes, indicating that padding was required during encryption. This can be explained by the fact that these are block cyphers, which

require that the plaintext be an exact multiple of the block length, so if that is not the case, padding is used to fill in the blanks.

- And I noticed that no padding was required in the cfb and ofb modes because the file size after decryption did not increase. This behaviour can be explained by the fact that these are stream cyphers, and streaming modes of operation can encrypt and decrypt messages of any size, so they do not require padding.

Task 5: Programming using the Crypto Library

So far, we have learned how to use the tools provided by openssl to encrypt and decrypt messages. In this task, we will learn how to use openssl's crypto library to encrypt/decrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in http://www.openssl.
org/docs/crypto/EVP_EncryptInit.html.
Activate MyEnv
Please get yourself familiar with this program, and then do the following exercise. You are given a plaintext and a ciphertext, and you know that aes-128-cbc is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character '0'). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. You can download a English word list from the Internet. We have also linked one on the web page of this lab. The plaintext and ciphertext is in the following:

```
Plaintext (total 21 characters): This is a top secret. Ciphertext (in hex format):
8d20e5056a8d24d0462ce74e4904c1b5
                                  13e10d1df4a2ef2ad4540fae1ca0aaf9
```

Words.txt file


```
preinstructs 24905113ddf1945ad8037a6c226deb420efcebc1741d37a84c09186e0689f27f --> NO MATCH
preinsula 9e37ddbb18185e596345bf36be7e2733067c3aead01c32d1e02eb2ad6e87f8b0 --> NO MATCH
scissor-tailed c40f8ca179cdd3d0eef551a792537bd3396f2f469e75a4d9091d055c6ed9eafc --> NO MATCH
scissor-winged e02fd5335f1455289782e4f52794d86971f0cbfff928417df34dc6d2a27969ad --> NO MATCH
scissorwise 838b6e46150007a1d0383f129c95c9e8b250e99203d483bf5e0704169e043492 --> NO MATCH
scissura 93031d7ebb39c3c20564052f6e0b2541e08e05e39001cf2f301e35e16e8fd919 --> NO MATCH
scissure 1715369cc97f5083ad9afa9bdecac5e8ffa985eba58cf33b163e503f6aadfa8a --> NO MATCH
zwiebel 080d15f65c1fb580a38e0f99e565869cc89fbb3773d4ad0f2e05422ab31c41ee --> NO MATCH
zwieselite 2f2342c4e1c99673b54aca30655a35d89767864c3f5e9718f90365d92360dfa0 --> NO MATCH
zwingle c99210ecf7014a9a948d5270ae6d3cd99fdafbc8cf5a68ca2b81877bea74f39b --> NO MATCH
zwingli b05d6713ea431a27239a46d191cadc9d5521c2ae25ec2596a8bc1dbf2bfab102 --> NO MATCH
zwinglian 5f2b18d9f93d7fa27fcca4c27581a2c272e864cf25a1e71652b86979ee6b5bac --> NO MATCH
zwinglianism aedf7feabae8a15a22334d5cd43844b60f946f0382d13a8acd60b84dd80348b0 --> NO MATCH
zwinglianist 17158ad1efe7d3e6fcd751fc2d801019d22b3f64a37e2ddf68ae9d877cf8a084 --> NO MATCH
zwitter 6df12d857a5b3461a67a6f4548250d705407c73c48cd5242f226ef38077d18b9 --> NO MATCH
zwitterion 0b794ab2da476a76aeab642ad9fe582dfd79cc0042113790d30adef2da8b51b5 --> NO MATCH
zwitterionic 43d9a3b874200bc70e02e6b0143c3e9495c296487814a3c4d4c8747d630c7de3 --> NO MATCH
zwolle 327589a13f6940f1b17e3f3e85826724e8b2bb8cebcecf4c151e7f91e625c635 --> NO MATCH
zworykin 770797ec4f4bd24b34e54ad3f7e958c05cc673631b47623b372e4473519e1305 --> NO MATCH
zz 5d7a15c6b07909667c79feac6346d16c02204f1c63389721179c2015cb21c2bb --> NO MATCH
zzt 58dae019524c7d7d79ceaa3a43186ce4a5f64d9ad1817ac4b06a649500de2ef8 --> NO MATCH
zzz 2bfc0c1cb86d489616d5eaf811512b493869e2b14dfe420eaba6a4a4f431e94b --> NO MATCH


Resulting Key: ['median']
```

Code

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

cipherHex = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"
plainText = b"This is a top secret."
result = []

file = open('words.txt', 'r')
lines = file.readlines()
words = [str.strip(line) for line in lines]
```

```
for word in words:
    if len(word) >= 16:
        continue
    word = word.lower()
    key = word.encode() + b' '*(16-len(word))
    cipher = AES.new(key, AES.MODE_CBC, iv=bytes.fromhex('0'*32))
    ciphertext = cipher.encrypt(pad(plainText, AES.block_size))
    is_matched = "--> NO MATCH"
    if bytes.hex(ciphertext) == cipherHex:
        is_matched = "--> MATCH"
        result.append(word)
    print(word, bytes.hex(ciphertext), is_matched)
print("\n\nResulting Key:",result)
```

Observations

The key is “median”

I discovered that by brute forcing the key using the pycryptodome module in Python using the given plain text, cypher text, and iv, I was able to find the key.

Conclusion

- In this experiment, I learned about various encryption methods such as AES and their different encryption modes such as ECB, CBC, CFB, OFB, etc and I implemented them using Openssl utility.
- ECB mode encrypts the identical plain text blocks to identical encrypted text blocks and hence it is less secure.
- From the experiment, I deduced that ECB and CBC employ padding while encrypting, whereas the other two don't. ECB and CBC are block cyphers, but CFB and OFB are stream cyphers, as evidenced by this.
- In OFB mode, if the single digit of the 30th byte corrupted, then in plain text that only that byte or character is corrupted. Thus, only OFB mode shows the most promising result in task 3 and almost all the texts are recovered.
- I concluded from task 5 that if I have the ciphertext and the key space, I can easily locate the plaintext using the brute force method.

Github Links

Repository

https://github.com/kashishvjain/CSS-Lab