

Secure Communications

Lab 4

Symmetric Key

Sections

A. OpenSSL

A.1 `systeminfo >> VolatileDataFile.txt` `openssl version`

```
[b00167321 kali] ~$ openssl list -cipher-cmds
aes-128-cbc    aes-128-ecb    aes-192-cbc    aes-192-ecb
aes-256-cbc    aes-256-ecb    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    aria-192-cfb1
aria-192-cfb8   aria-192-ctr    aria-192-ecb    aria-192-ofb
aria-256-cbc    aria-256-cfb    aria-256-cfb1   aria-256-cfb8
aria-256-ctr    aria-256-ecb    aria-256-ofb    bf-nofb
bf-cbc          bf-cfb        bf-ecb        bf-nofb
camellia-128-cbc camellia-128-ecb camellia-192-cbc camellia-192-ecb
camellia-256-cbc camellia-256-ecb cast           cast-cbc
cast5-cbc       cast5-cfb     cast5-ecb      cast5-ofb
des             des-cbc       des-cfb       des-ecb
des-edc         des-edc-cbc   des-edc-cfb   des-edc-ofb
des-ed3         des-ed3-cbc   des-ed3-cfb   des-ed3-ofb
des-ofb         des3          desx          rc2
rc2-40-cbc     rc2-40-cbc   rc2-cbc      rc2-cfb
rc2-ecb         rc2-ofb      rc4           rc4-40
seed            seed-cbc     seed-cfb      seed-ecb
seed-Ofb        seed-Ofb     seed-Ofb      seed-Ofb
sm4             sm4-cbc     sm4-cfb      sm4-ctr

[b00167321 kali] ~$ openssl version
OpenSSL 3.5.0 8 Apr 2025 (Library: OpenSSL 3.5.0 8 Apr 2025)
[b00167321 kali] ~$
```

— Outline five encryption methods that are supported:

→ **aes-128-cbc aria-128-cbc bf-cbc camellia-128-cbc des-cbc**

— Outline the version of OpenSSL:

→ **OpenSSL 3.5.0**

```
A2openssl prime -hex 1111 openssl prime 42 openssl prime 1421

└─(b00167321# kali)-[~]
└─$ openssl prime -hex 1111
1111 (1111) is not prime

└─(b00167321# kali)-[~]
└─$ openssl prime 42
29 (42) is not prime

└─(b00167321# kali)-[~]
└─$ openssl prime 1421
58D (1421) is not prime

└─(b00167321# kali)-[~]
└─$
```

— Check if the following are prime numbers:

- 1111 (1111) is not prime
 - 2A (42) is not prime
 - 58D (1421) is not prime

```
A.3 openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin

[+] b00167321# kali㉿kali:~
[+] $ echo "secret" > myfile.txt

[+] b00167321# 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.

[+] b00167321# kali㉿kali:~
[+] $ cat encrypted.bin
Salted__myXZ0+=====
[+] b00167321# kali㉿kali:~
[+] $
```

— Is it easy to write out or transmit the output:

- No

A.4 `openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64`

```
[b00167321# kali㉿~]
└─$ echo "secret" > myfile.txt
[b00167321# 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.
[b00167321# kali㉿~]
└─$ cat encrypted.bin
U2FsdGVkX1B91Y1ZSMuMfAmkSq1a/KCs9iMGonNB6FI=
[b00167321# kali㉿~]
└─$
```

— Is it easy to write out or transmit the output:

→ Yes

A.5 `openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64`

```
[b00167321# kali㉿~]
└─$ echo "secret" > myfile.txt
[b00167321# 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.
[b00167321# kali㉿~]
└─$ cat encrypted.bin
U2FsdGVkX1B91Y1ZSMuMfAmkSq1a/KCs9iMGonNB6FI=
[b00167321# 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.
[b00167321# kali㉿~]
└─$ cat encrypted.bin
U2FsdGVkX1BC06gpxhspp3lCsrjTqkYLc1iGFunXCr4=
[b00167321# kali㉿~]
└─$
```

— Has the output changed?

→ Yes

— Why has it changed?

→ Because OpenSSL uses a random initialization vector (IV) for each encryption, ensuring different ciphertexts even for identical inputs.

A.6 `openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:napier -base64`

```
[+] $ openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:napier -base64
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
secret
[+] $ -
```

— Has the output been decrypted correctly?

→ Yes

— What happens when you use the wrong password?

→ Decryption fails

```
[+] $ openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:napier -base64
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
bad decrypt
4057431FCB779E00:error:1C800064:Provider routines:ossi_cipher_nupadblock:bad decrypt:./providers/implementations/ciphers/ciphercommon_block.c:107:
```

A.7 `openssl enc -bf-cbc -in myfile.txt -out encrypted.bin`

`openssl enc -d -bf-cbc -in encrypted.bin -out decrypted.txt`

```
[+] $ echo "secret" > myfile.txt
[+] $ cat myfile.txt
secret
[+] $ b00167321# kali)-(~]
[+] $ openssl enc -bf-cbc -in myfile.txt -out encrypted.bin
enter BF-CBC encryption password:
Verifying - enter BF-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[+] $ cat encrypted.bin
Salted__*****Z#
[+] $ b00167321# kali)-(~]
[+] $ openssl enc -d -bf-cbc -in encrypted.bin -out decrypted.txt
enter BF-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[+] $ cat decrypted.txt
secret
[+] $
```

— Did you manage to decrypt the you can decrypt it. file?

→ Yes

B. Padding (AES)

B.1 With AES which uses a 256-bit key, what is the normal...

 bill's
A security site.com
+ profsims.com - Networksims

Welcome to Asecuritysite.com

[HOME](#) | [INDEX](#) | [CIPHER](#) | [BLOGS](#) | [IP](#) | [IDS](#) | [MAGIC](#) | [NET](#) | [CISCO](#) | [CYBER](#) | [TEST](#) | [FUN](#) | [SUBJ](#) | [ABOUT](#)

Padding

[Encryption Home][Home]

Padding is used in a block cipher where we fill up the blocks with padding bytes. AES uses 128-bits (16 bytes), and DES uses 64-bit blocks (8 bytes). The main padding methods are:

- CMS (Cryptographic Message Syntax). This pads with the same value as the number of padding bytes. Defined in RFC 5652, PKCS#5, PKCS#7 (X.509 certificate) and RFC 1423 PEM.
 - Bit. This pads with 0x80 (10000000) followed by zero (null) bytes. Defined in ANSI X.923 and ISO/IEC 9797-1.
 - ZeroLength. This pads with zeros except for the last byte which is equal to the number (length) of padding bytes.
 - Null. This pads will NULL bytes. This is only used with ASCII text.
 - Space. This pads with spaces. This is only used with ASCII text.
 - Random. This pads with random bytes with the last byte defined by the number of padding bytes.

— Block size (bytes):

→ 16 bytes (128-bits)

— Number of hex characters for block size:

→ **32** (1 byte = 2 hex chars → $16 \times 2 = 32$)

B.2 message - "kettle" password - "oxtail"

= CMS:

→ 61e145 (61e14593c46cf5747d2c0c7fe78a3ec8)

= Null:

→ **18eff3** (18eff3cb081f1319c517a25992fc246b)

= Space:

→ 8e460b (8e460b692e3158d04e85cef7db6aeb2e)

B.3 How many hex characters will be used for the 256-bit AES encryption

```
In this implementation, only CMS has been implemented
After padding (CMS): 66f7874726f74090909090909090909
Cipher (ECB): ea8e0d5147c3eda4d3cecc4f06ef60652
[6c4NUUFD7aT7zsTwbyYGUg==]
decrypt: foxtrot
```

```
In this implementation, only CMS has been implemented
After padding (CMS): 66f7874726f74616e74656174657201
Cipher (ECB): 7947a050f4abeeeefdf6d295aed3c10fc
[eUegUF9Kvu7fbSl17TwQ/A=]
decrypt: foxtrotanteater
```

— Number of hex characters:

- “**fox**”: 32 hex chars (3 bytes) → 1 block
 - “**foxtrot**”: 32 hex chars (7 bytes) → 1 block
 - “**foxtrotanteater**”: 32 hex chars (15 bytes) → 1 block
 - “**foxtrotanteatercastle**”: 64 hex chars (21 bytes) → $\lceil 21/16 \rceil = 2$ blocks

B.4 With 256-bit AES, for n characters in a string, generalise the calculation of the number of...

— Hex characters:

- $$\rightarrow 2 \times (16 \times \text{ceil}(n / 16)) = 32 \times \text{ceil}(n / 16)$$

— Base-64 characters:

- $$\begin{aligned}\rightarrow 4 \times \text{ceil}(b / 3) &= 4 \times \text{ceil}((16 \times \text{ceil}(n / 16)) / 3) \\ \rightarrow b &= 16 \times \text{ceil}(n / 16)\end{aligned}$$

C. Padding (AES)

C.1 With DES which uses a 64-bit key, what is the normal...

The screenshot shows the Asecuritysite.com website. The header features a logo of a man in a top hat and bow tie, the text "bill's A security site.com + profsims.com - Networksims", and a "Welcome to Asecuritysite.com" message. Below the header is a navigation bar with links: HOME, INDEX, CIPHER, BLOGS, IP, IDS, MAGIC, NET, CISCO, CYBER, TEST, FUN, SUBJ, and ABOUT. The main content area has a title "Padding (DES)" with a small icon. Below it is a red banner with "[Encryption Home][Home]". The main text discusses padding methods for DES, listing CMS, Bit, ZeroLength, Null, Space, and Random padding. To the right of the text is a cartoon character of a person with curly hair.

Padding is used in a block cipher where we fill up the blocks with padding bytes. DES uses 64-bits with a 64-bit encryption key. The main padding methods are:

- CMS. This pads with the same value as the number of padding bytes. Defined in RFC 5652, PKCS#5, PKCS#7 and RFC 1423 PEM.
- Bit. This pads with 0x80 (10000000) followed by zero (null) bytes. Defined in ANSI X.923 and ISO/IEC 9797-1.
- ZeroLength. This pads with zeros except for the last byte which is equal to the number (length) of padding bytes.
- Null. This pads with NULL bytes. This is only used with ASCII text.
- Space. This pads with spaces. This is only used with ASCII text.
- Random. This pads with random bytes with the last byte defined by the number of padding bytes.

— Block size (bytes):

→ **8 bytes (64-bits)**

— Number of hex characters for block size:

→ **16 (1 byte = 2 hex chars → $8 \times 2 = 16$)**

C.2 message - "kettle" password - "oxtail"

```
In this implementation, only CMS has been implemented
DES
After padding (CMS): 6b6574746c650202
Cipher (ECB): 0d74d0510d32caaa
[DXTQUQ0yyqo=]
decrypt: kettle

After padding (Bit): 6b6574746c650202
Cipher (ECB): 0d74d0510d32caaa
[DXTQUQ0yyqo=]
decrypt: kettle

After padding (ZeroLen): 6b6574746c650202
Cipher (ECB): 0d74d0510d32caaa
[DXTQUQ0yyqo=]
decrypt: kettle

After padding (Null): 6b6574746c650202
Cipher (ECB): 0d74d0510d32caaa
[DXTQUQ0yyqo=]
decrypt: kettle0000000000000000

After padding (Space): 6b6574746c650000
Cipher (ECB): 8400ede37908c60c
[HADt43kIxgw=]
decrypt: kettle0000000000000000

After padding (Random): 6b6574746c650202
Cipher (ECB): 0d74d0510d32caaa
[DXTQUQ0yyqo=]
decrypt: kettle
```

— CMS:

→ **0d74d0 (0d74d0510d32caaa)**

— Null:

→ **0d74d0 (0d74d0510d32caaa)**

— Space:

→ **8400ed (8400ede37908c60c)**

C.3 How many hex characters will be used for the 256-bit AES encryption

```
In this implementation, only CMS has been implemented  
DES  
After padding (CMS): 666f780505050505  
Cipher (ECB): 9e9a68b4cecef3fd  
[nppoth708/0=]  
decrypt: fox
```

```
In this implementation, only CMS has been implemented  
DES  
After padding (CMS): 666f7874726f7401  
Cipher (ECB): 85af0285d8bcc6a1  
[ha8Chdi8xqB=]  
decrypt: foxtrot
```

```
In this implementation, only CMS has been implemented  
DES  
After padding (CMS): 666f7874726f74616e74656174657201  
Cipher (ECB): 0d2780ea10b35444e39aac485977c13b8  
[DSeA6HczVETjmssF13wTuA==]  
decrypt: foxtrotanteater
```

```
In this implementation, only CMS has been implemented  
DES  
After padding (CMS): 666f7874726f74616e746561746572636173746c65030303  
Cipher (ECB): 0d2780ea10b354449ebc0eb89aac5c790276358ebde5127a  
[DSeA6HczVESevA64mgxceQJ2NY695RJ6]  
decrypt: foxtrotanteatercastle
```

— Number of hex characters:

- “**fox**”: **16 hex chars** (3 bytes) → 1 block
- “**foxtrot**”: **16 hex chars** (7 bytes) → 1 block
- “**foxtrotanteater**”: **32 hex chars** (15 bytes) → 2 block
- “**foxtrotanteatercastle**”: **48 hex chars** (21 bytes) → 3 blocks

C.4 With 64-bit DES, for n characters in a string, generalise the calculation of the number of...

— Hex characters:

$$\rightarrow 2 \times (8 \times \text{ceil}(n / 8)) = 16 \times \text{ceil}(n / 8)$$

— Base-64 characters:

$$\rightarrow 4 \times \text{ceil}(b / 3) = 4 \times \text{ceil}((8 \times \text{ceil}(n / 8)) / 3)$$

$$\rightarrow b = 8 \times \text{ceil}(n / 8)$$

Lab 2: Symmetric Key

Objective: The key objective of this lab is to understand the range of symmetric key methods used within symmetric key encryption. We will introduce block ciphers, stream ciphers and padding. The key tools used include OpenSSL, Python and JavaScript. Overall Python 2.7 has been used for the sample examples, but it should be easy to convert these to Python 3.x.

□ Web link (Weekly activities): <https://asecuritysite.com/esecurity/unit02>

Demo: <https://youtu.be/N3UADaXmOik>

A OpenSSL

OpenSSL is a standard tool that we used in encryption. It supports many of the standard symmetric key methods, including AES, 3DES and ChaCha20.

No	Description	Result
A.1	Use: openssl l list-cipher-methods openssl version	Outline five encryption methods that are supported: aes-256-cbc aes-128-cbc des-cbc cast128-128-cbc des-SSLE
		Outline the version of OpenSSL: OpenSSL 1.3.5-x
A.2	Using openssl and the command in the form: openssl prime -hex 1111	Check if the following are prime numbers: 42 [Yes] [No] 1421 [Yes] [No]
A.3	Now create a file named myfile.txt (either use Notepad or another editor). Next encrypt with aes-256-ebc openssl enc -aes-256-cbc -in myFile.txt -out encrypted.bin and enter your password.	Use the following command to view the output file: cat encrypted.bin Is it easy to write out or transmit the output: [Yes] [No]
A.4	Now repeat the previous command and add the -base64 option. openssl enc -aes-256-cbc -in myFile.txt -out encrypted.bin -base64	Use following command to view the output file: cat encrypted.bin Is it easy to write out or transmit the output: [Yes] [No]
A.5	Now Repeat the previous command and observe the encrypted output.	Has the output changed? [Yes] [No]

1

	<code>openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64</code>	Why has it changed? Because we used a random initialization vector (IV) for each encryption, meaning different ciphertexts will result in different keys.
A.6	Now let's decrypt the encrypted file with the correct format:	Has the output been decrypted correctly? <input checked="" type="checkbox"/>
	<code>openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:napiyer -base64</code>	What happens when you use the wrong password? <input checked="" type="checkbox"/> Encryption failed
A.7	Now encrypt a file with Blowfish and see if you can decrypt it.	Did you manage to decrypt the file? <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No

B Padding (AES)

With encryption, we normally use a block cipher, and where we must pad the end blocks to make sure that the data fits into a whole number of block. Some background material is here:

■ **Web link (Padding):** <http://asecuritysite.com/encryption/padding>

In the first part of this tutorial we will investigate padding blocks:

No	Description	Result
B.1	With AES which uses a 256-bit key, what is the normal block size (in bytes).	Block size (bytes): 16 bytes (128-bits) Number of hex characters for block size: 32 <small>(16 bytes * 2 hex characters = 32 hex = 32)</small>
B.2	Go to: Web link (AES Padding): http://assecurecitysite.com/encryption/padding	CMS: 44646 Null: 44677 Space: 44660
B.3	Using 256-bit AES encryption, and a message of "kettle" and a password of "oxtaill", determine the cipher using the differing padding methods (you only need to show the first six hex characters). If you like, copy and paste the Python code from the page, and run it on your Kali instance.	

Lock

		44 has char 0x00 0x00 0x00 0x00
B.4	With 256-bit AES, for n characters in a string, how would you generalise the calculation of the number of hex characters in the cipher text.	“fotrottanteaste”
	How many Base-64 characters would be used (remember 6 bits are used to represent a Base-64 character);	<p>Hex characters: $2 \times (\lfloor n \times \text{charlen} / 3 \rfloor + \lfloor (n \times \text{charlen}) \% 3 \rfloor)$ </p> <p>Base-64 characters:</p> <pre> * $\lfloor n \times \text{charlen} / 3 \rfloor$ * $\lfloor (n \times \text{charlen}) \% 3 \rfloor$ * $n - 16 \times \text{ceil}(n / 16)$ </pre>

C Padding (DES)

In the first part of this lab we will investigate padding blocks:

No	Description	Result
C.1	With DES which uses a 64-bit key, what is the normal block size (in bytes)?	Block size (bytes): 8 bytes (64-bits)
C.2	Go to: Web link (DES Padding): http://assecureto.com/encryption/padding_des	Number of hex characters for block size: 16 (1 byte = 2 hex chars, 8x 2 = 16)
C.3	Using 64-bit DES key encryption, and a message of “kettle” and a password of “oxtail”, determine the cipher using the differing padding methods.	CMS: 8d74685156c32caee Null: 8d74685156c32caee Space: 444400e77980c6dc
C.4	If you like, copy and paste the Python code from the copy, and run it on your Kali instance.	If you like, copy and paste the Python code from the copy, and run it on your Kali instance.
C.5	For the following words, estimate how many hex characters will be used for the 64-bit key DES encryption:	Number of hex characters: “fox”: 16 hex chars (1 byte = 2 hex chars) “fotxtrot”: 16 hex chars (1 byte = 2 hex chars) “fotxtrotanteated”: 32 hex chars (1 byte = 2 hex chars) “fotxtrotanteatecastile”: 64 hex chars (1 byte = 2 hex chars)
C.6	With 64-bit DES, for n characters in a string, how would you generalise the calculation of the number of hex characters in the cipher text.	Hex characters: ? x 16 = 2 * n + 8 (1 byte = 2 hex chars, 8 bytes = 16 hex chars) Base-64 characters: 4 * ceil(n / 3) + 8 = ceil(4n / 3) + 8 (1 byte = 2 hex chars, 3 bytes = 6 hex chars)

3

How many Base-64 characters would be used (remember 6 bits are used to represent a Base-64 character):

D Python Coding (Encrypting)

In this part of the lab, we will investigate the usage of Python code to perform different padding methods and using AES. First download the code from:

Web link (Cipher code): <http://asecuritysite.com/cipher01.zip>

The code should be:

```

from Crypto.Cipher import AES
import hashlib
import sys
from binascii import hexlify, unhexlify
import Padding
from struct import pack, unpack

class AESCipher:
    def __init__(self, key):
        self.key = hashlib.sha256(key).digest()

    def encrypt(self, plaintext, mode):
        if mode == "ECB":
            plaintext = Padding.appendPadding(plaintext, self.blocksize)
            return ECB_MODE_ecb.encrypt(unhexlify(plaintext))
        elif mode == "CBC":
            ciphertext = ECB_MODE_ecb.encrypt(unhexlify(key))
            plaintext = ECB_MODE_ecb.decrypt(ciphertext) + hexlify(plaintext)
            return ECB_MODE_ecb.encrypt(unhexlify(plaintext))

    def decrypt(self, ciphertext, mode):
        if mode == "ECB":
            ciphertext = ECB_MODE_ecb.decrypt(unhexlify(ciphertext))
            return ECB_MODE_ecb.decrypt(ciphertext)
        elif mode == "CBC":
            ciphertext = ECB_MODE_ecb.decrypt(unhexlify(ciphertext))
            return ECB_MODE_ecb.decrypt(ciphertext)

key = hashlib.sha256("password").digest()

plainText = Padding.appendPadding("attack", self.blocksize)
print plainText

ciphertext = encrypt(ciphertext, key, AES_MODE_ECB)
print "Cipher : " + hexlify(ciphertext)

plaintext = decrypt(ciphertext, key, AES_MODE_ECB)
print "Plaintext : " + hexlify(plaintext)

print "Decrypted : " + plaintext

plainText = decrypt(ciphertext, key, AES_MODE_CBC)
print plainText

```

Now update the code so that you can enter a string and the program will show the cipher text. The format will be something like:

```
python cipher01.py hello mykey
```

where "hello" is the plain text, and "mykey" is the key. A possible integration is

```
import sys  
if (len(sys.argv)>1):  
    val=sys.argv[1]  
if (len(sys.argv)>2):  
    password=sys.argv[2]
```

Now determine the cipher text for the following (the first example has already been completed):