Secure Communications

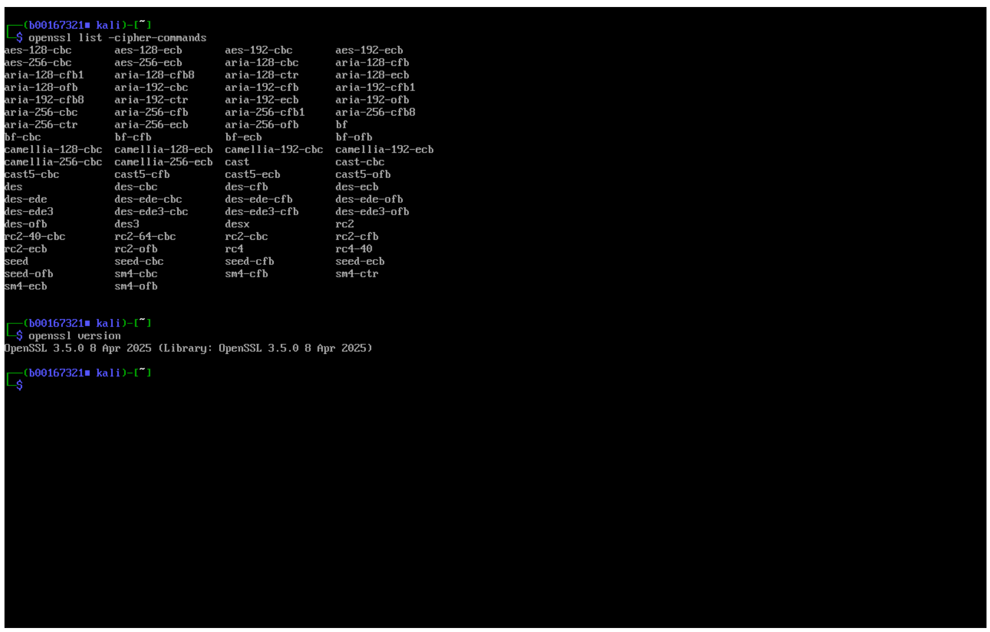
Week 5

Symmetric Key (Part 1)

Sections

A. OpenSSL

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



— 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**

A.2 `openssl prime -hex 1111` `openssl prime 42` `openssl prime 1421`

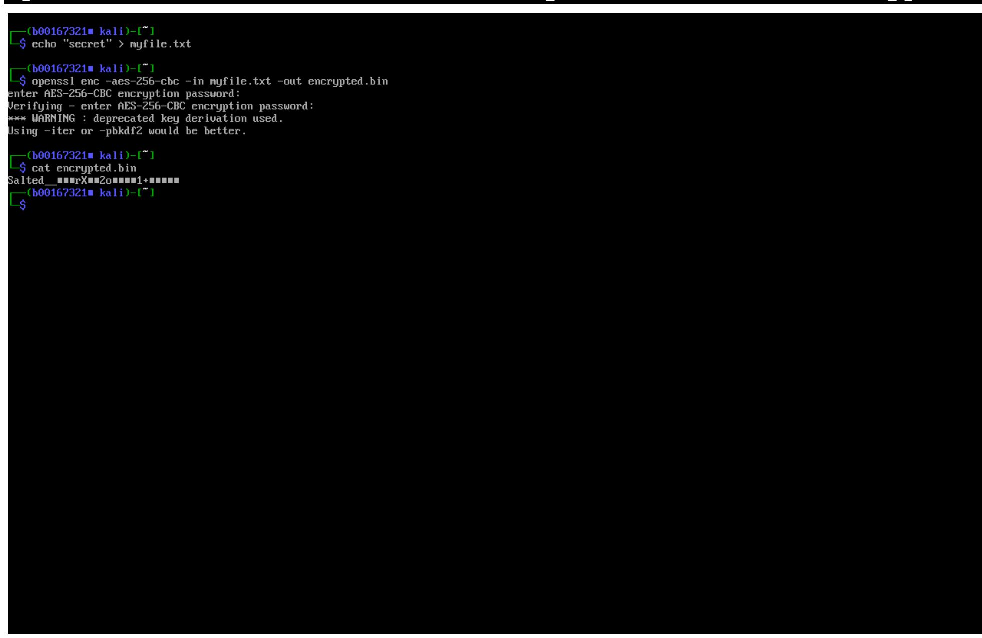
— Check if the following are prime numbers:

→ **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`



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

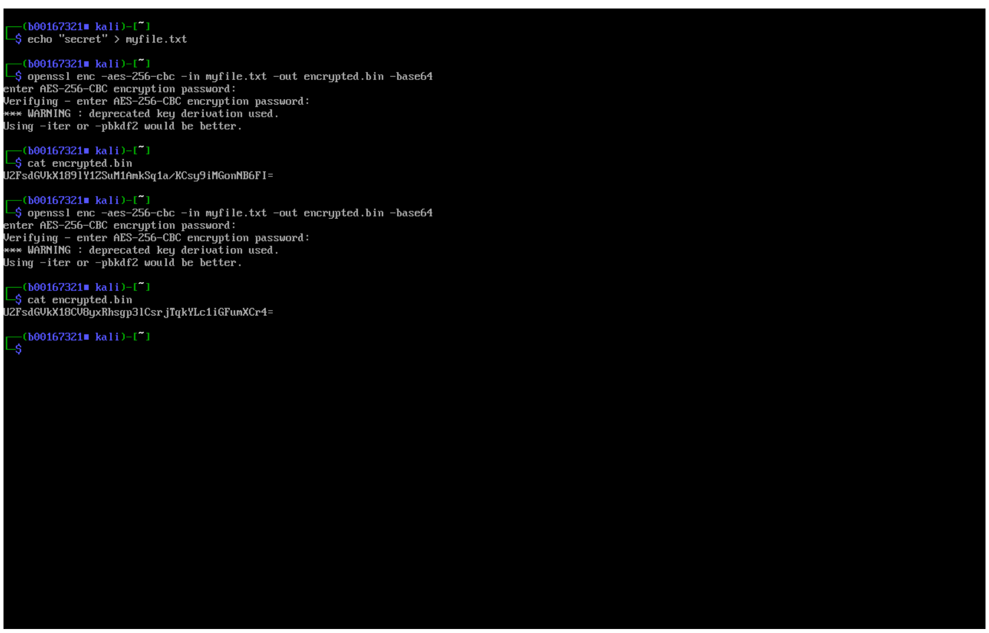
→ **No**

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

<u>— Is it easy to write out or transmit the output:</u>

→ **Yes**

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



<u>— Has the output changed?</u>

→ **Yes**

<u>— Why has it changed?</u>

→ **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`

<u>— Has the output been decrypted correctly?</u>

→ **Yes**

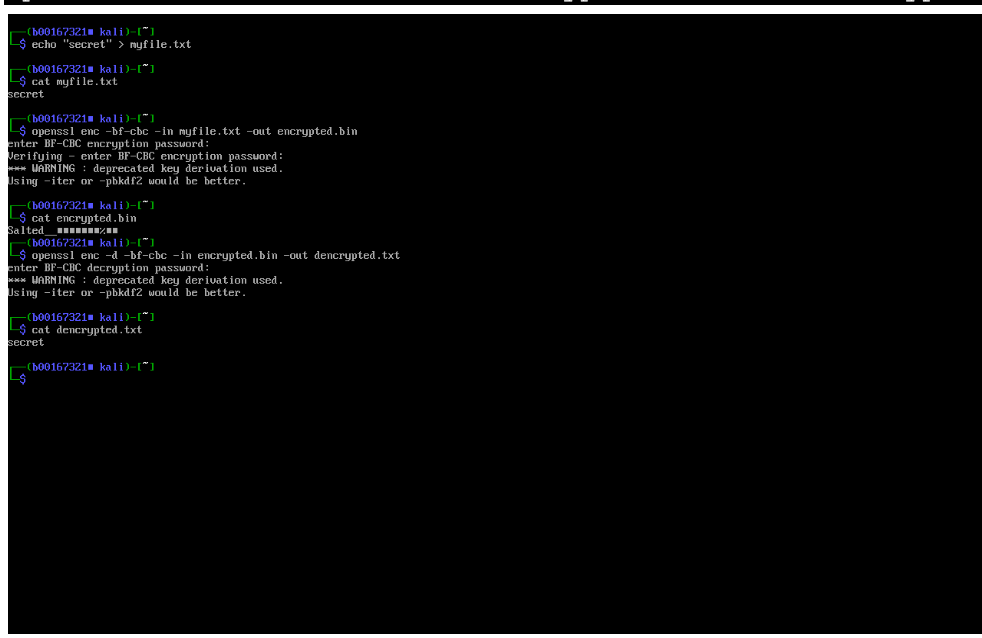
<u>— What happens when you use the wrong password?</u>

→ **Decryption fails**

→ 

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

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

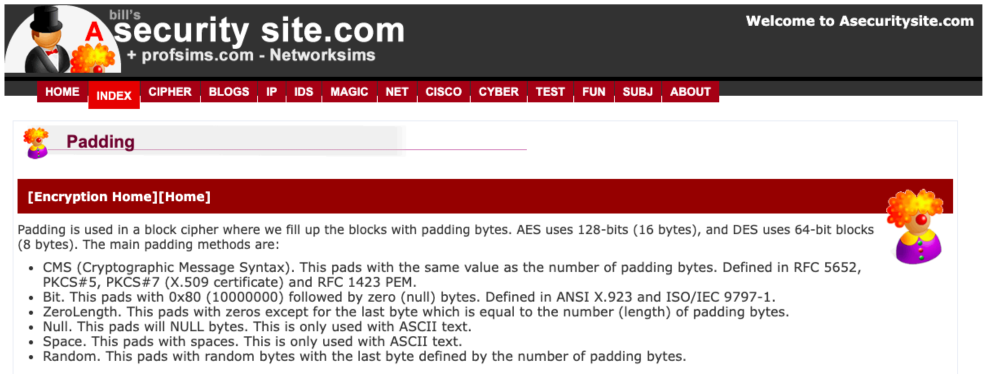


<u>— Did you manage to decrypt the you can decrypt it. file?</u>

→ **Yes**

B. Padding (AES)

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



— Block size (bytes):

→ **16 bytes (128-bits)**

— Number of hex characters for block size:

→ **32** (1 byte = 2 hex chars → 16×2 = 32)

B.2 `message – "kettle"` `password – "oxtail"`

```
In this implementation, only CMS has been implemented
After padding (CMS): 6b6574746c650a0a0a0a0a0a0a0a0a0a
Cipher (ECB): 61e14593c46cf5747d2c0c7fe78a3ec8
  [YeFFk8Rs9XR9LAx/54o+yA==]
  decrypt: kettle

After padding (Bit): 6b6574746c6580000000000000000000
Cipher (ECB): cc6203662893bdb75c008a31d0953860
  [zGIDZiiTvbdcAIox0JU4YA==]
  decrypt: kettle

After padding (ZeroLen): 6b6574746c6500000000000000000009
Cipher (ECB): b6e70dd15ba0e98e5e745bd9b3c02827
  [tucN0Vug6Y5edFvZs8AoJw==]
  decrypt: kettle

After padding (Null): 6b6574746c6500000000000000000000
Cipher (ECB): 18eff3cb081f1319c517a25992fc246b
  [GO/zywgfExnFF6JZkvwkaw==]
  decrypt: kettle

After padding (Space): 6b6574746c6520202020202020202020
Cipher (ECB): 8e460b692e3158d04e85cef7db6aeb2e
  [jkYLaS4xWNBOhc7322rrLg==]
  decrypt: kettle

After padding (Random): 6b6574746c6553678797a80249605309
Cipher (ECB): 96d70454d894498bffb921a7ee9c21fa
  [ltcEVNiUSYv/uSGn7pwh+g==]
  decrypt: kettle
```

— 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): 666f780d0d0d0d0d0d0d0d0d0d0d0d0d
Cipher (ECB): 961068ddb012e940efe96e033f517cfd
  [lhBo3bAS6UDv6W4DP1F8/Q==]
  decrypt: fox
```

```
In this implementation, only CMS has been implemented
After padding (CMS): 666f7874726f74090909090909090909
Cipher (ECB): ea8e0d5147c3eda4d3cec4f06ef60652
  [6o4NUUfD7aTTzsTwbvYGUg==]
  decrypt: foxtrot
```

```
In this implementation, only CMS has been implemented
After padding (CMS): 666f7874726f74616e74656174657201
Cipher (ECB): 7947a0505f4abeeedf6d295aed3c10fc
  [eUegUF9Kvu7fbSla7TwQ/A==]
  decrypt: foxtrotanteater
```

```
In this implementation, only CMS has been implemented
After padding (CMS): 666f7874726f74616e7465617465726361737 46c650b0b0b0b0b0b0b0b0b0b0b
Cipher (ECB): 69590213ee77a52938ab6370f1f02a65a8344a83393552683cccde0b6b049ea5
  [aVkCE+53pSk4q2Nw8fAqZag0SoM5NVJoPMzeC2sEnqU=]
  decrypt: foxtrotanteatercastle
```

— 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) → ceil(21/16)=2 blocks

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

— Hex characters:

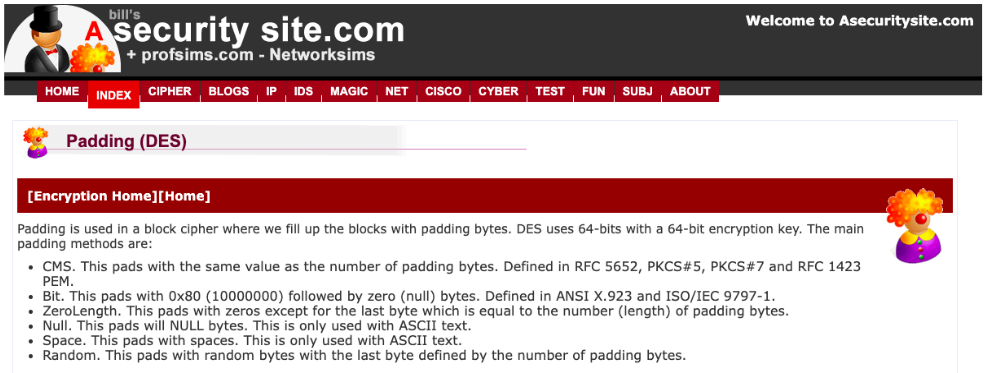
- → **2 × (16 × ceil(n / 16)) = 32 × ceil(n / 16)**

— Base-64 characters:

- → **4 × ceil(b / 3) = 4 × ceil((16 × ceil(n / 16)) / 3)**
- → b = 16 × ceil(n / 16)

C. Padding (AES)

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



— <u>Block size (bytes):</u>

→ **8 bytes (64-bits)**

— <u>Number of hex characters for block size:</u>

→ **16** (1 byte = 2 hex chars → 8×2 = 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: kettle

After padding (Space): 6b6574746c650000
Cipher (ECB): 8400ede37908c60c
  [hADt43kIxgw=]
  decrypt: kettle����������

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

— <u>CMS:</u>

→ **0d74d0** (0d74d0510d32caaa)

— <u>Null:</u>

→ **0d74d0** (0d74d0510d32caaa)

— <u>Space:</u>

→ **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
  [nppotM7O8/0=]
  decrypt: fox
```

```
In this implementation, only CMS has been implemented
DES
After padding (CMS): 666f7874726f7401
Cipher (ECB): 85af0285d8bcc6a1
  [ha8Chdi8xqE=]
  decrypt: foxtrot
```

```
In this implementation, only CMS has been implemented
DES
After padding (CMS): 666f7874726f74616e74656174657201
Cipher (ECB): 0d2780ea10b35444e39ac485977c13b8
  [DSeA6hCzVETjmsSFl3wTuA==]
  decrypt: foxtrotanteater
```

```
In this implementation, only CMS has been implemented
DES
After padding (CMS): 666f7874726f74616e746561746572636173746c65030303
Cipher (ECB): 0d2780ea10b354449ebc0eb89aac5c790276358ebde5127a
  [DSeA6hCzVESevA64mqxceQJ2NY695RJ6]
  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:

- → **2 × (8 × ceil(n / 8)) = 16 × ceil(n / 8)**

— Base-64 characters:

- → **4 × ceil(b / 3) = 4 × ceil((8 × ceil(n / 8)) / 3)**
- → b = 8 × 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 list-cipher-commands`
	
	`openssl version`
		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

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-cbc `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]

	`openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64`	Why has it changed? Because OpenSSL uses a random initialization vector (IV) for each encryption, ensuring different ciphertexts even for identical inputs.
A.6	Now let's decrypt the encrypted file with the correct format: `openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:napier -base64`	Has the output been decrypted correctly? Yes What happens when you use the wrong password? Decryption fails
A.7	Now encrypt a file with Blowfish and see if you can decrypt it.	Did you manage to decrypt the file? [Yes][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 (1 byte = 2 hex chars → 16×2 = 32)
B.2	Go to: 🕮 **Web link (AES Padding):** `http://asecuritysite.com/encryption/padding` Using 256-bit AES encryption, and a message of "kettle" and a password of "oxtail", 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.	CMS: 61e145 Null: 18eff3 Space: 8e460b
B.3	For the following words, estimate how many hex characters will be used for the 256-bit AES encryption:	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) → ceil(21/16)=2 blocks
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. How many Base-64 characters would be used (remember 6 bits are used to represent a Base-64 character):	Hex characters: 2 × (16 × ceil(n / 16)) = 32 × ceil(n / 16) Base-64 characters: 4 × ceil(b / 3) = 4 × ceil((16 × ceil(n / 16)) / 3) b = 16 × ceil(n / 16)

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) Number of hex characters for block size: 16 (1 byte = 2 hex chars → 8×2 = 16)
C.2	Go to: 🕮 **Web link (DES Padding):** http://asecuritysite.com/encryption/padding_des Using 64-bit DES key encryption, and a message of "kettle" and a password of "oxtail", determine the cipher using the differing padding methods. If you like, copy and paste the Python code from the page, and run it on your Kali instance.	CMS: 0d74d0510d32caaa Null: 0d74d0510d32caaa Space: 8400ede37908c60c
C.3	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 (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 block
C.4	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: 2 × (8 × ceil(n / 8)) = 16 × ceil(n / 8) Base-64 characters: 4 × ceil(b / 3) = 4 × ceil((8 × ceil(n / 8)) / 3) b = 8 × ceil(n / 8)

	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
import binascii
import Padding

val='hello'
password='hello'

plaintext=val

def encrypt(plaintext,key, mode):
        encobj = AES.new(key,mode)
        return(encobj.encrypt(plaintext))

def decrypt(ciphertext,key, mode):
        encobj = AES.new(key,mode)
        return(encobj.decrypt(ciphertext))

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


plaintext = Padding.appendPadding(plaintext,blocksize=Padding.AES_blocksize,mode='CMS')
print "After padding (CMS): "+binascii.hexlify(bytearray(plaintext))

ciphertext = encrypt(plaintext,key,AES.MODE_ECB)
print "Cipher (ECB): "+binascii.hexlify(bytearray(ciphertext))

plaintext = decrypt(ciphertext,key,AES.MODE_ECB)
plaintext = Padding.removePadding(plaintext,mode='CMS')
print "  decrypt: "+plaintext


plaintext=val
```

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):