Secure Communications

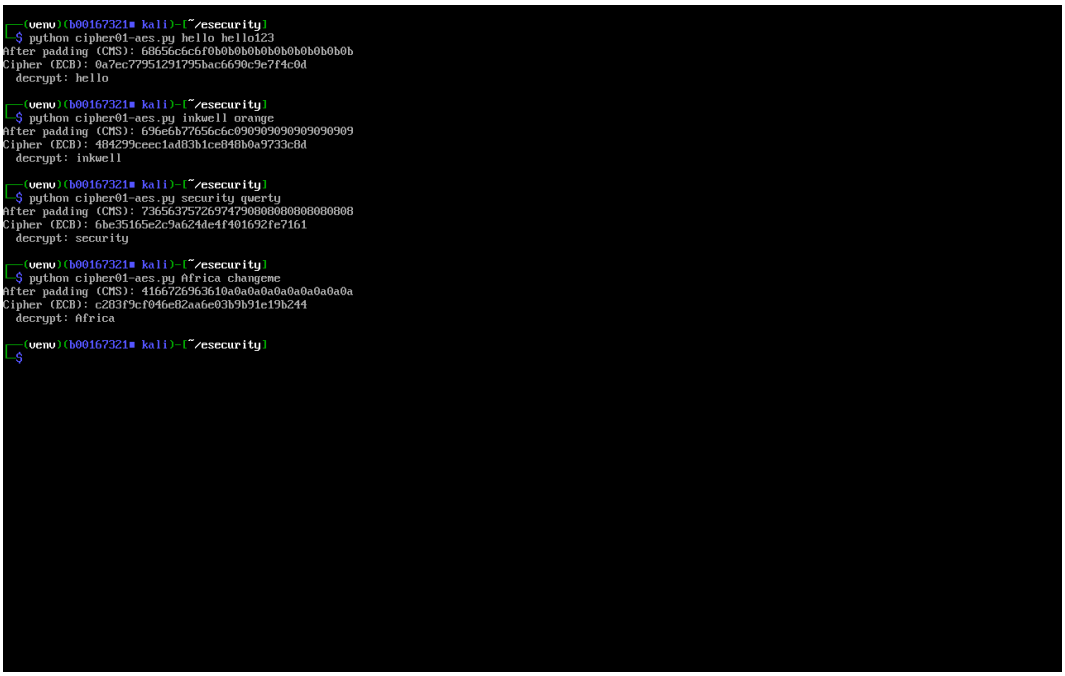
Lab 5

Symmetric Key (Part 2)

Sections

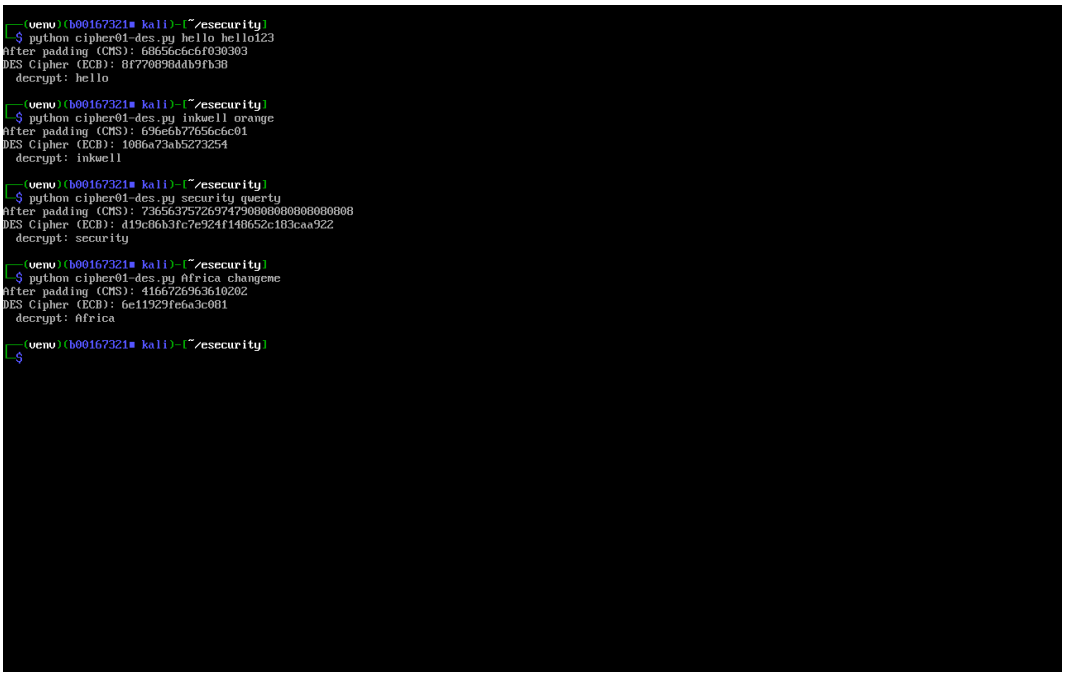
D. Python Coding (Encrypting)

D.1 `python cipher01-aes.py (AES)`



Message	**Key**	**CMS Cipher**
“hello”	“hello123”	0a7ec77951291795bac6690c9e7f4c0d
“inkwell”	“orange”	484299ceec1ad83b1ce848b0a9733c8d
“security”	“qwerty”	6be35165e2c9a624de4f401692fe7161
“Africa”	“changeme”	c283f9cf046e82aa6e03b9b91e19b244

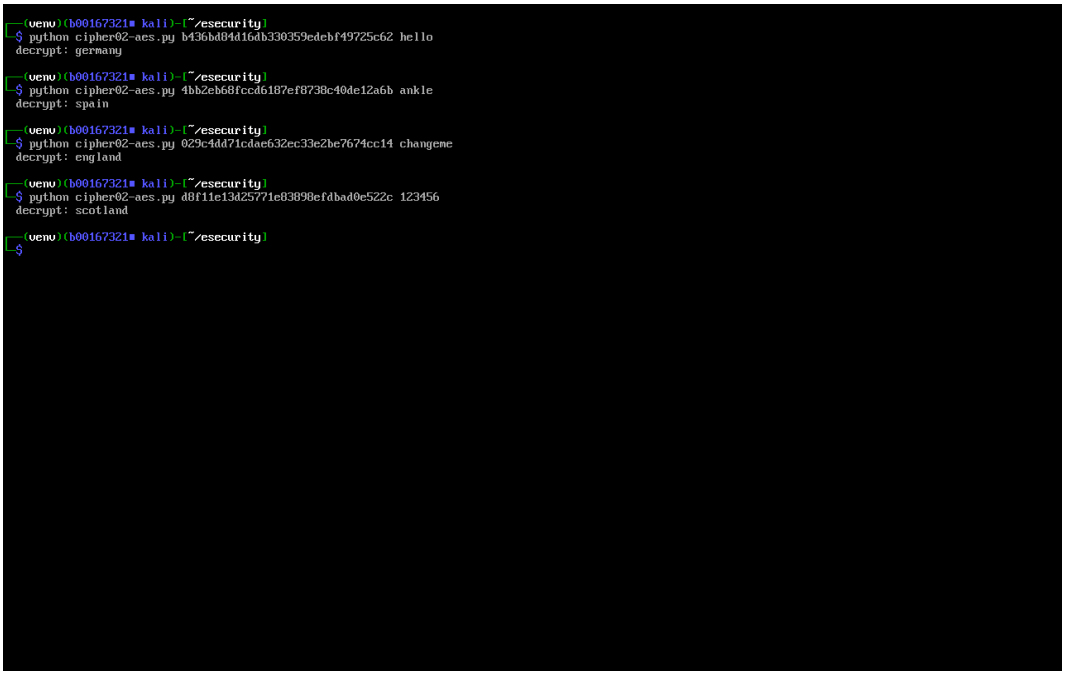
D.2 `python cipher01-des.py (DES)`



Message	**Key**	**CMS Cipher**
“hello”	“hello123”	8f770898ddb9fb38
“inkwell”	“orange”	1086a73ab5273254
“security”	“qwerty”	d19c86b3fc7e924f148652c183caa922
“Africa”	“changeme”	6e11929fe6a3c081

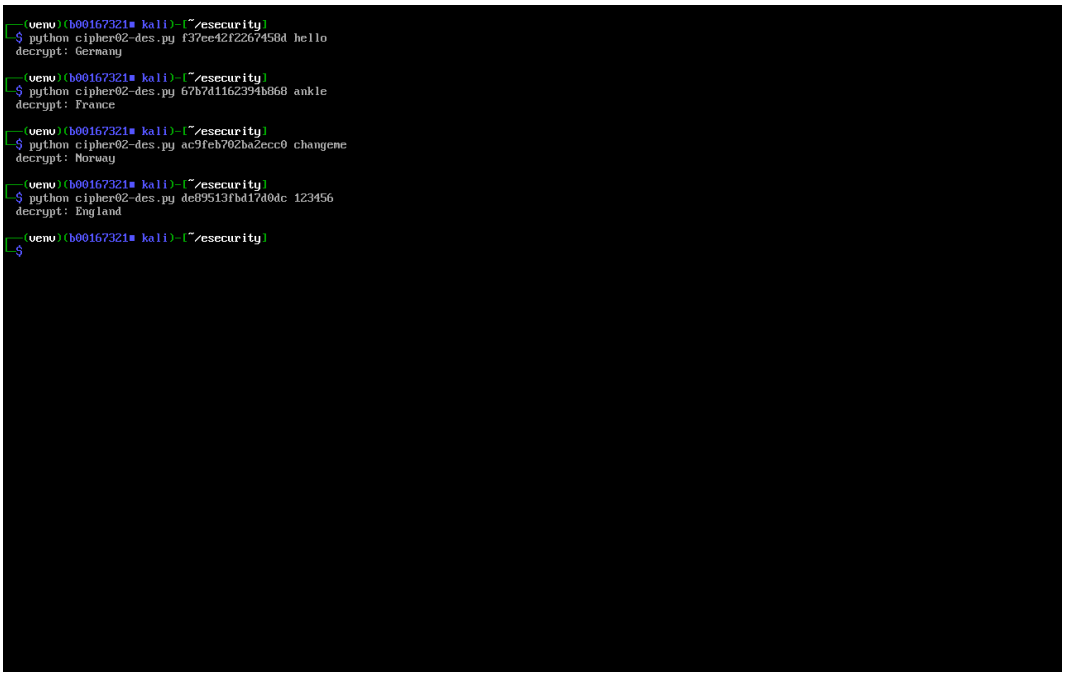
E. Python Coding (Decrypting)

E.1 `python cipher02-aes.py (AES)`



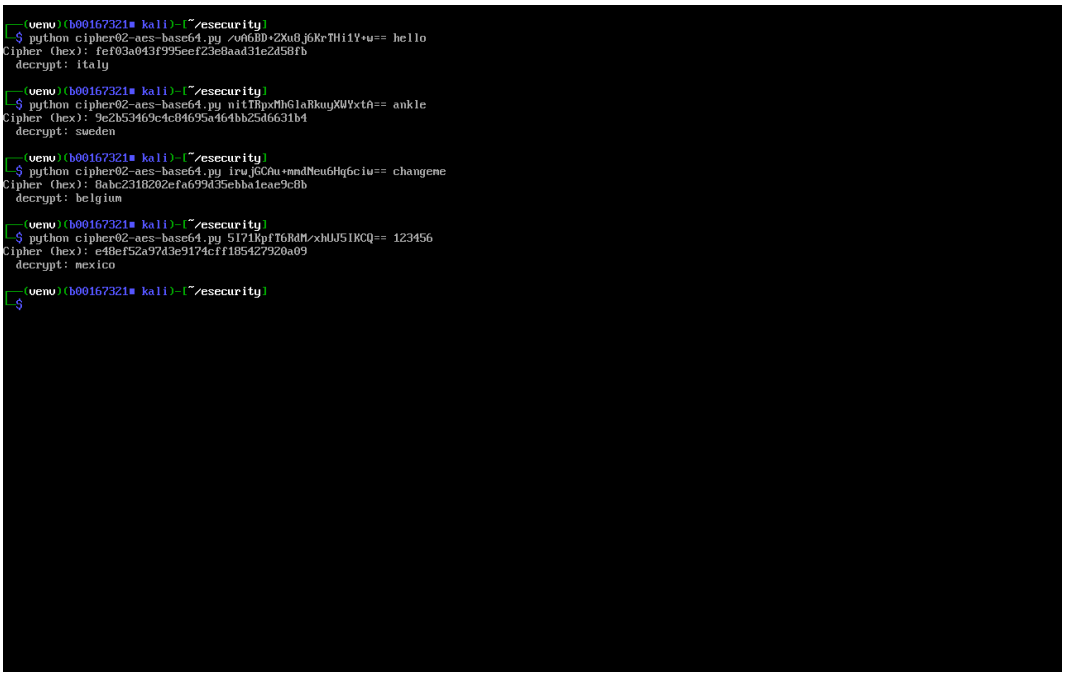
CMS Cipher (256-bit AES ECB)	**Key**	**Plain text**
b436bd84d16db330359edebf49725c62	“hello”	germany
4bb2eb68fccd6187ef8738c40de12a6b	“ankle”	spain
029c4dd71cdae632ec33e2be7674cc14	“changeme”	england
d8f11e13d25771e83898efdbad0e522c	“123456”	scotland

E.2 `python cipher02-des.py (DES)`



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

E.3 `python cipher02-aes-base64.py (AES 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

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 - 16x2 = 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: `8e468b`
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` "foxtrotantеatercastle": `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 - 8x2 = 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: `0d74d0510d32caan` Null: `0d74d0510d32caan` Space: `0400ede37988c60c`
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. How many Base-64 characters would be used (remember 6 bits are used to represent a Base-64 character):	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)`

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):

Message	Key	CMS Cipher
"hello"	"hello123"	0a7ec77951291795bac6690c9c7f4c0d
"inkwell"	"orange"	484299ceec1ad83b1ce848b0a9733c8d
"security"	"qwerty"	6be35165e2c9a624de4f401692fe7161
"Africa"	"changeme"	c283f9cf646e82aa6e83b9b91e19b24d

Now copy your code and modify it so that it implements **64-bit DES** and complete the table (Ref to: http://asecuritysite.com/encryption/padding_des):

Message	Key	CMS Cipher
"hello"	"hello123"	8f770898ddb9fb38
"inkwell"	"orange"	1086a73ab2272254
"security"	"qwerty"	d19c86b3fc7e924f148652c183caa922
"Africa"	"changeme"	6e11929fe6a3c081

Now modify the code so that the user can enter the values from the keyboard, such as with:

```
cipher=raw_input('Enter cipher:')
password=raw_input('Enter password:')
```

E Python Coding (Decrypting)

Now 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 (note, all the plain text values are countries around the World):

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

Now 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 (note, all the plain text values are countries around the World):

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

Now 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 (which should give countries of the World):

CMS Cipher (256-bit AES ECB)	Key	Plain text
/vA6BD+ZXu8j6KrTH1IY+w==	"hello"	Italy
n1tTRpxMhGlaRkuyXWYxtA==	"ankle"	Sweden
1rwJGCAu+mmdNeu6Hq6c1w==	"changeme"	Belgium
5i71KpfT6RdM/xhUJ5IKCQ==	"123456"	Mexico

PS … remember to add "`import base64`".

Notes

The code can be downloaded from:

```
git clone https://github.com/billbuchanan/esecurity
```

If you need to update the code, go into the esecurity folder, and run:

```
git pull
```

To install a Python library use:

```
pip install libname
```

To install a Node.js package, use:

```
npm install libname
```

For B.2 you might need to install these:

```
pip install pycrypt
pip install padding
```