Symmetric Key and Hashing

Objective: The key objective of this lab is to investigate the basics of symmetric key encryption.

1   AES (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.encode()).digest()


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

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

plaintext = decrypt(ciphertext,key,AES.MODE_ECB)

plaintext = Padding.removePadding(plaintext.decode(),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"	0a7ec77951291795bac6690c9e7f4c0d
"inkwell"	"orange"	
"security"	"qwerty"	
"Africa"	"changeme"	

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

cipher=input('Enter cipher:')

password=input('Enter password:')

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"	
4bb2eb68fccd6187ef8738c40de12a6b	"ankle"	
029c4dd71cdae632ec33e2be7674cc14	"changeme"	
d8f11e13d25771e83898efdbad0e522c	"123456"	

PS … remember to add "import base64".

2   Hashing

MD5 and SHA-1 produce a hash signature, but this can be attacked by rainbow tables. Bcrypt (Blowfish Crypt) is a more powerful hash generator for passwords and uses salt to create a non-recurrent hash. It was designed by Niels Provos and David Mazières, and is based on the Blowfish cipher. It is used as the default password hashing method for BSD and other systems.

Overall it uses a 128-bit salt value, which requires 22 Base-64 characters. It can use a number of iterations, which will slow down any brute-force cracking of the hashed value. For example, "Hello" with a salt value of "$2a$06$NkYh0RCM8pNWPaYvRLgN9." gives:

$2a$**06**$**NkYh0RCM8pNWPaYvRLgN9.**LbJw4gcnWCOQYIom0P08UEZRQQjbfpy

As illustrated in Figure 1, the first part is "$2a$" (or "$2b$"), and then followed by the number of rounds used. In this case is it **6 rounds** which is 2^6 iterations (where each additional round

doubles the hash time). The 128-bit (22 character) salt values comes after this, and then finally there is a 184-bit hash code (which is 31 characters).

The slowness of bcrypt is highlighted with an AWS EC2 server benchmark using hashcat:

- Hash type: MD5 Speed/sec: 380.02M words
- Hash type: SHA1 Speed/sec: 218.86M words
- Hash type: SHA256 Speed/sec: 110.37M words
- Hash type: bcrypt, Blowfish(OpenBSD) Speed/sec: 25.86k words
- Hash type: NTLM. Speed/sec: 370.22M words

You can see that Bcrypt is almost 15,000 times slower than MD5 (380,000,000 words/sec down to only 25,860 words/sec). With John The Ripper:

- md5crypt [MD5 32/64 X2] 318237 c/s real, 8881 c/s virtual
- bcrypt ("$2a$05", 32 iterations) 25488 c/s real, 708 c/s virtual
- LM [DES 128/128 SSE2-16] 88090K c/s real, 2462K c/s virtual

where you can see that BCrypt over 3,000 times slower than LM hashes. So, although the main hashing methods are fast and efficient, this speed has a down side, in that they can be cracked easier. With Bcrypt the speed of cracking is considerably slowed down, with each iteration doubling the amount of time it takes to crack the hash with brute force. If we add one onto the number of rounds, we double the time taken for the hashing process. So, to go from 6 to 16 increase by over 1,000 (2^{10}) and from 6 to 26 increases by over 1 million (2^{20}).

The following defines a Python script which calculates a whole range of hashes:

```
import hashlib;
import passlib.hash;


salt="ZDzPE45C"
string="password"
salt2="111111111111111111111"


print ("General Hashes")
print ("MD5:"+hashlib.md5(string.encode()).hexdigest())
print ("SHA1:"+hashlib.sha1(string.encode()).hexdigest())
print ("SHA256:"+hashlib.sha256(string.encode()).hexdigest())
print ("SHA512:"+hashlib.sha512(string.encode()).hexdigest())


print ("UNIX hashes (with salt)")
print ("DES:"+passlib.hash.des_crypt.encrypt(string.encode(), salt=salt[:2]))
```

```
print ("MD5:"+passlib.hash.md5_crypt.encrypt(string.encode(), salt=salt))

print ("Sun MD5:"+passlib.hash.sun_md5_crypt.encrypt(string.encode(), salt=salt))

print ("SHA1:"+passlib.hash.sha1_crypt.encrypt(string.encode(), salt=salt))

print ("SHA256:"+passlib.hash.sha256_crypt.encrypt(string.encode(), salt=salt))

print ("SHA512:"+passlib.hash.sha512_crypt.encrypt(string.encode(), salt=salt))
```
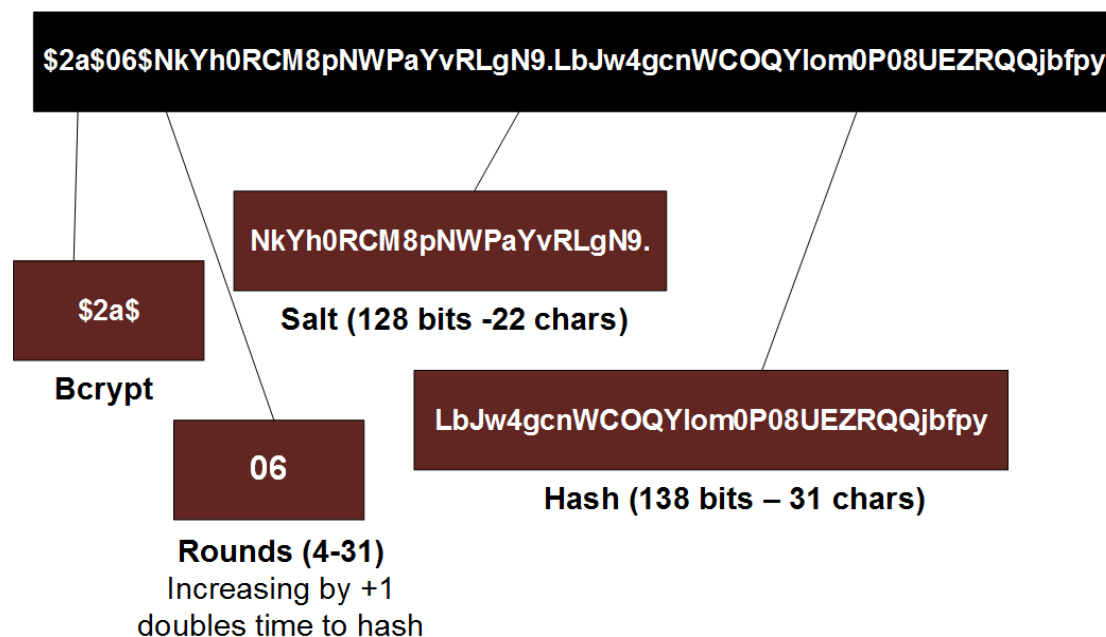
$2a$06$NkYh0RCM8pNWPaYvRLgN9.LbJw4gcnWCOQYlom0P08UEZRQQjbfpy

NkYh0RCM8pNWPaYvRLgN9.

Salt (128 bits -22 chars)

$2a$

Bcrypt

LbJw4gcnWCOQYlom0P08UEZRQQjbfpy

Hash (138 bits – 31 chars)

06

Rounds (4-31)
Increasing by +1
doubles time to hash

Figure 1 Bcrypt

No	Description	Result
H.1	Create the hash for the word "hello" for the different methods (you only have to give the first six hex characters for the hash): Also note the number hex characters that the hashed value uses:	MD5: SHA1: SHA256: SHA512: DES: MD5: Sun MD5: SHA-1: SHA-256: SHA-512: