Lab 2.5: Padding

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:

http://asecuritysite.com/encryption/padding

A demo of this tutorial is here: https://youtu.be/kghv6q1xv_o

A Padding (AES)

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

No	Description	Result
1	With AES which uses a 256-bit key, what is the normal block size (in bytes).	Block size (bytes): Number of hex characters for block size:
2	Go to: 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).	CMS: Bit: ZeroLength: Null: Space: Random:
3	For the following words, estimate how many hex characters will be used for the 256-bit AES encryption:	Number of hex characters: "fox": "foxtrot": "foxtrotanteater": "foxtrotanteatercastle":
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.	Hex characters: Base-64 characters:

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

B Padding (DES)

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

No	Description	Result
1	With DES which uses a 64-bit key, what is the normal block size (in bytes):	Block size (bytes): Number of hex characters for block size:
2	Go to: http://asecuritysite.com/encryption/padding Using 64-bit DES encryption, and a message of "kettle" and a password of "oxtail", determine the cipher using the differing padding methods.	CMS: Bit: ZeroLength: Null: Space: Random:
3	For the following words, estimate how many hex characters will be used for the 256-bit AES encryption:	Number of hex characters: "fox": "foxtrot": "foxtrotanteater": "foxtrotanteatercastle":
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: Base-64 characters:

C 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:

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


plaintext = Padding.appendPadding(plaintext,blocksize=Padding.AES_blocksize,mode='ZeroLen')
print "\nAfter padding (Bit): "+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,blocksize=Padding.AES_blocksize,mode='ZeroLen')
print "  decrypt: "+plaintext


plaintext=val

plaintext = Padding.appendPadding(plaintext,blocksize=Padding.AES_blocksize,mode='Space')
print "\nAfter padding (Null): "+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,blocksize=Padding.AES_blocksize,mode='Space')
print "  decrypt: "+plaintext


plaintext=val

plaintext = Padding.appendPadding(plaintext,blocksize=Padding.AES_blocksize,mode='Random')
print "\nAfter padding (Random): "+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='Random')
print "  decrypt: "+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=int(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 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"	db8d4d4ddaea314f
"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=raw_input('Enter cipher:')
password=raw_input('Enter password:')
```

D 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”	
4bb2eb68fccd6187ef8738c40de12a6b	“ankle”	
029c4dd71cdae632ec33e2be7674cc14	“changeme”	
d8f11e13d25771e83898efdbad0e522c	“123456”	

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
2305f602668cc1bed46f6b30a46e7e77	“hello”	
c1e201682b46ac3c	“ankle”	
77533df9d91c2b3f	“changeme”	
0aba553d307646d1	“123456”	

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+ZXu8j6KrTHi1Y+w==	“hello”	
nitTRpxMhGlaRkuyXWYxtA==	“ankle”	
irwjGCAu+mmdNeu6Hq6ciw==	“changeme”	
5I71KpfT6RdM/xhUJ5IKCQ==	“123456”	

PS … remember to add “using base64”

E Catching exceptions

If we try “1jDmCTD1IfbXbyyHgAyrdg==” with “hello”, we should get a country. What happens when we try the wrong key:

Output when we use “hello”:

Output when we use “hello1”:

Now catch the exception with an exception handler:

```
try:
        plaintext = Padding.removePadding(plaintext,mode='CMS')
        print "  decrypt: "+plaintext
except:
        print("Error!")
```

Now implement a Python program which will try various keys for a cipher text input, and show the decrypted text. The keys tried should be:

["hello","ankle","changeme","123456"]

Run the program and try to crack:

“1jDmCTD1IfbXbyyHgAyrdg==”

What is the password:

F Reflective questions

1. If we have five ‘a’ values (“aaaaa”). What will be the padding value used for 256-bit AES with CMS:

2. If we have six ‘a’ values (“aaaaaa”). What will be the hex values used for the plain text:

3. For the different padding methods, when will the cipher stream be the same for each of the different methods? Give an example of a phrase which creates the same cipher text for each of the other padding methods:

4. The following cipher text is 256-bit AES ECB for a number of spaces (0x20):

```
c3f791fad9f9392116b2d12c8f6c4b3dc3f791fad9f9392116b2d12c8f6c4b
3dc3f791fad9f9392116b2d12c8f6c4b3dc3f791fad9f9392116b2d12c8f6c
4b3da3c788929dd8a9022bf04ebf1c98a4e4
```

What can you observe from the cipher text:

What is the range that is possible for the number of spaces which have been used:

How might you crack a stream sequence like this:

Answers

A possible answer for Section D is:

```
from Crypto.Cipher import AES
import hashlib
import sys
import binascii
import Padding

val='hello'
password='hello'


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

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()


ciphertext = binascii.unhexlify(cipher)

print "Cipher (ECB): "+binascii.hexlify(bytearray(ciphertext))

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

A sample code for Section E is:

```
pw = ["hello","ankle","changeme","123456"]

for password in pw:

      try:
            key = hashlib.sha256(password).digest()
            cipherhex = base64.decodestring(cipher).encode('hex')

            ciphertext = binascii.unhexlify(cipherhex)

            print "Cipher (ECB): "+binascii.hexlify(bytearray(ciphertext))

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

      except:
            print(".")
```