Lab 5: Diffie-Hellman and Public Key

Demo: http://youtu.be/3n2TMpHqE18

1 Diffie-Hellman

No	Description	Result
1	Bob and Alice have agreed on the values: g=2,879, N= 9,929 Bob Select b=6, Alice selects a=9	Now calculate (using the Kali calculator): Bob's B value (g^b mod N): Alice's A value (g^a mod N):
2	Now they exchange the values. Next calculate the shared key:	Bob's value (A^b mod N): Alice's value (B^a mod N): Do they match? [Yes] [No]
3	If you are in the lab, select someone to share a value with. Next agree on two numbers (g and N). You should pick a random number, and so should they. Do not tell them what your random number is. Next calculate your public value, and get them to do the same. Next exchange values.	Numbers for g and N: Your private value: Your public value: The public value you received: Shared key: Do they match: [Yes] [No]

2 Public Key

No	Description	Result
1	With RSA, we have a public modulus (and which is N=p.q, and where p and q are prime numbers). To create this, we need to generate a key pair with: `openssl genrsa -out private.pem 1024` This file contains both the public and the private key.	What is the type of public key method used: How long is the default key: How long are the prime numbers that are used to generate the public key?
2	Use following command to view the output file: `cat private.pem`	What can be observed at the start and end of the file:
3	Next we view the RSA key pair: `openssl rsa -in private.pem -text -noout` You should now see the public exponent (e), the private exponent (d), the two prime numbers (p and q), and the public modulus (N).	Which number format is used to display the information on the attributes: Which are the elements of the key shown: Which are the elements of the public key? Which are the elements of the private key? What does the –noout option do?
4	Let's now secure the encrypted key with 3-DES: `openssl rsa -in private.pem -des3 -out key3des.pem`	

	You should NEVER share your private key.	
5	Next, we will export the public key: `openssl rsa -in private.pem -out public.pem -outform PEM -pubout`	View the output public key. What does the header and footer of the file identify? Is the public key smaller in size than the private key? [Yes/No]
6	Now we will encrypt with our public key: `openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin`	
7	And then decrypt with our private key: `openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt`	What are the contents of decrypted.txt:
8	If you are working in the lab, now give your public key to your neighbour, and get them to encrypt a secret message for you.	Did you manage to decrypt their message? [Yes][No]

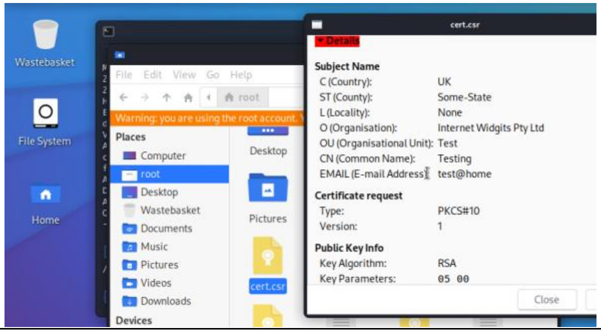
3 Storing keys

We have stored our keys on a key ring file (PEM). Normally we would use a digital certificate to distribute our public key. In this part of the tutorial we will create a crt digital certificate file.

No	Description	Result
1	Next create the crt file with the following:	View the CRT file by double clicking on it from the File Explorer.

<table>
<tr>
<td></td>
<td>

<code>openssl req -new -key private.pem -out cert.csr</code>



</td>
<td>

Using Google to search, what is PKCS#10, and which is it used for?

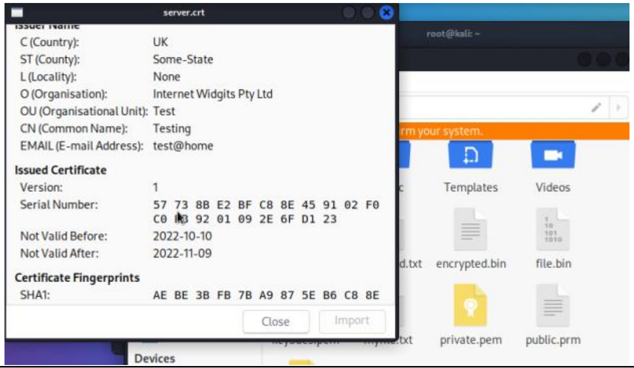
What is the type of public key method used:

What is the public key size (in bits): [512][<u>1024</u>][2048]

</td>
</tr>
<tr>
<td></td>
<td>

We can now take the code signing request, and create a certificate. For this we sign the certificate with a private key, in order to validate it:

<code>openssl x509 -req -in cert.csr -signkey private.pem -out server.crt</code>



</td>
<td>

From the File System, click on the newly created certificate file (server.crt) and determine:

The size of the public key (in bits): [512][<u>1024</u>][2048]

The public key encryption method:

Which is the hashing method that has been signed to sign the certificate: [MD5][<u>SHA-1</u>][SHA-256]

</td>
</tr>
</table>

5 Python tutorial

In Python, we can use the Hazmat (Hazardous Materials) library to implement symmetric key encryption.

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

The code should be:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend

import hashlib
import sys
import binascii

val='hello'
password='hello123'

plaintext=val

def encrypt(plaintext,key, mode):
    method=algorithms.AES(key)
    cipher = Cipher(method,mode, default_backend())
    encryptor = cipher.encryptor()
    ct = encryptor.update(plaintext) + encryptor.finalize()
    return(ct)

def decrypt(ciphertext,key, mode):
    method=algorithms.AES(key)
    cipher = Cipher(method, mode, default_backend())
    decryptor = cipher.decryptor()
    pl = decryptor.update(ciphertext) + decryptor.finalize()
    return(pl)

def pad(data,size=128):
    padder = padding.PKCS7(size).padder()
    padded_data = padder.update(data)
    padded_data += padder.finalize()
    return(padded_data)

def unpad(data,size=128):
    padder = padding.PKCS7(size).unpadder()
    unpadded_data = padder.update(data)
```

```
    unpadded_data += padder.finalize()
    return(unpadded_data)

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

print("Before padding: ",plaintext)

plaintext=pad(plaintext.encode())

print("After padding (CMS): ",binascii.hexlify(bytearray(plaintext)))

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

plaintext = decrypt(ciphertext,key,modes.ECB())

plaintext = unpad(plaintext)
print("  decrypt: ",plaintext.decode())
```

How is the encryption key generate?

Which is the size of the key used? [128-bit][256-bit]

Which is the encryption mode used? [ECB][CBC][OFB]

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”	

Finally, change the program so that it does 256-bit AES with CBC mode.