

# Lab 5: Diffie-Hellman, Public Key, Private Key and Hashing

Part 1 Demo: <http://youtu.be/3n2TMpHqE18>

The hashcat version has a time-out, so enter the following command:

```
date -s "1 OCT 2015 18:00:00"
```

## 1 Diffie-Hellman

No	Description	Result
1	On Kali, login and get an IP address using:  <code>sudo dhclient eth0</code>	What is your IP address?
2	Bob and Alice have agreed on the values:  $G=2879$ , $N=9929$ Bob Select $x=6$ , Alice selects $y=9$	Now calculate (using the Kali calculator):  Bob's A value ( $G^x \bmod N$ ):  Alice's B value ( $G^y \bmod N$ ):
3	Now they exchange the values. Next calculate the shared key:	Bob's value ( $B_x \bmod N$ ):  Alice's value ( $A_y \bmod N$ ):  Do they match? [Yes] [No]
4	If you are in the lab, select someone to share a value with. Next agree on two numbers (G and N).	Numbers for G and N:  Your x value:

	<p>You should generate a random number, and so should they. Do not tell them what your random number is. Next calculate your A value, and get them to do the same.</p> <p>Next exchange values.</p>	<p>Your A value:</p> <p>The B value you received:</p> <p>Shared key:</p> <p>Do they match: [Yes] [No]</p>
--	---	---

## 2 Private Key

No	Description	Result
1	<p>Use:</p> <pre>openssl list-cipher-commands</pre> <pre>openssl version</pre>	<p>Outline five encryption methods that are supported:</p> <p>Outline the version of OpenSSL:</p>
2	<p>Using openssl and the command in the form:</p> <pre>openssl prime -hex 1111</pre>	<p>Check if the following are prime numbers:</p> <p>42 [Yes][No]</p> <p>1421 [Yes][No]</p>
3	<p>Now create a file named myfile.txt (either use Notepad or another editor).</p> <p>Next encrypt with aes-256-cbc</p> <pre>openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin</pre>	<p>Use following command to view the output file:</p> <pre>cat encrypted.bin</pre> <p>Is it easy to write out or transmit the output: [Yes][No]</p>

	and enter your password.	
<b>4</b>	<p>Now repeat the previous command and add the <code>-base64</code> option.</p> <pre>openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64</pre>	<p>Use following command to view the output file:</p> <pre>cat encrypted.bin</pre> <p>Is it easy to write out or transmit the output: [Yes][No]</p>
<b>5</b>	<p>Now repeat the previous command and observe the encrypted output.</p> <pre>openssl enc -aes-256-cbc -in myfile.txt -out encrypted.bin -base64</pre>	<p>Has the output changed? [Yes][No]</p> <p>Why has it changed?</p>
<b>6</b>	<p>Now let's decrypt the encrypted file with the correct format:</p> <pre>openssl enc -d -aes-256-cbc -in encrypted.bin -pass pass:<i>napier</i> -base64</pre>	<p>Has the output been decrypted correctly?</p> <p>What happens when you use the wrong password?</p>
<b>7</b>	<p>If you are working in the lab, now give your secret passphrase to your neighbour, and get them to encrypt a secret message for you.</p> <p>To receive a file, you listen on a given port (such as Port 1234)</p> <pre>nc -l -p 1234 &gt; enc.bin</pre> <p>And then send to a given IP address with:</p> <pre>nc -w 3 [IP] 1234 &lt; enc.bin</pre>	<p>Did you manage to decrypt their message? [Yes][No]</p>

### 3 Public Key

---

No	Description	Result
1	<p>First we need to generate a key pair with:</p> <pre>openssl genrsa -out private.pem 1024</pre> <p>This file contains both the public and the private key.</p>	<p>What is the type of public key method used:</p> <p>How long is the default key:</p> <p>How long did it take to generate a 1,024 bit key?</p> <p>View the contents of the keys.</p>
2	<p>Use following command to view the output file:</p> <pre>cat private.pem</pre>	<p>What can be observed at the start and end of the file:</p>
3	<p>Next we view the RSA key pair:</p> <pre>openssl rsa -in private.pem -text -noout</pre>	<p>Which are the attributes of the key shown:</p> <p>Which number format is used to display the information on the attributes:</p> <p>What does the <code>-noout</code> option do?</p>

<b>4</b>	Let's now secure the encrypted key with 3-DES:  openssl rsa -in private.pem -des3 -out key3des.pem	
<b>5</b>	Next we will export the public key:  openssl rsa -in private.pem -out public.pem -outform PEM -pubout	View the output key. What does the header and footer of the file identify?
<b>6</b>	Now we will encrypt with our public key:  openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin	
<b>7</b>	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
<b>8</b>	If you are working in the lab, now give your password to your neighbour, and get them to encrypt a secret message for you.	Did you manage to decrypt their message? [Yes][No]

## 4 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	<p>Next create the crt file with the following:</p> <pre>openssl req -new -key private.pem -out cert.csr</pre> <pre>openssl x509 -req -in cert.csr -signkey private.pem -out server.crt</pre>	<p>View the CRT file by double clicking on it from the File Explorer.</p> <p>What is the type of public key method used:</p> <p>View the certificate file and determine:</p> <p>The size of the public key:</p> <p>The encryption method:</p>

## 5 Hashing

<http://youtu.be/Xvbk2nSzEPk>

No	Description	Result
1	<p>Using:</p> <p><a href="http://asecuritysite.com/encryption/md5">http://asecuritysite.com/encryption/md5</a></p> <p>Match the hash signatures with their words (“Falkirk”, “Edinburgh”, “Glasgow” and “Stirling”).</p> <pre>03CF54D8CE19777B12732B8C50B3B66F D586293D554981ED611AB7B01316D2D5 48E935332AADEC763F2C82CDB4601A25 EE19033300A54DF2FA41DB9881B4B723</pre>	<p>03CF5: Is it [Falkirk][Edinburgh][Glasgow][Stirling]?</p> <p>D5862: Is it [Falkirk][Edinburgh][Glasgow][Stirling]?</p> <p>48E93: Is it [Falkirk][Edinburgh][Glasgow][Stirling]?</p> <p>EE190: Is it [Falkirk][Edinburgh][Glasgow][Stirling]?</p>

2	<p>Using:</p> <p><a href="http://asecuritysite.com/encryption/md5">http://asecuritysite.com/encryption/md5</a></p> <p>Determine the number of hex characters in the following hash signatures.</p>	<p>MD5 hex chars:</p> <p>SHA-1 hex chars:</p> <p>SHA-256 hex chars:</p> <p>How does the number of hex characters relate to the length of the hash signature:</p>
3	<p>Kali, for the following /etc/shadow file, determine the matching password:</p> <pre>bill:\$apr1\$waZS/8Tm\$jDZmiZBct/c2hysERCZ3m1 mike:\$apr1\$mKfrJquI\$Kx0CL9krmqhCu0SHKqp5Q0 fred:\$apr1\$Jbe/hCIb\$/k3A4kjpJyC06BUUaPRks0 ian:\$apr1\$0GyPhsLi\$jTTzw0HNS4C15ZEoyFLjB. jane: \$1\$rq0IRBBN\$R2pOQH9egTTVN1N1st2U7.</pre>	<p>The passwords are password, napier, inkwell and Ankle123. [Hint: openssl passwd -apr1 -salt ZaZS/8TF napier]</p> <p>Bill's password:</p> <p>Mike's password:</p> <p>Fred's password:</p> <p>Ian's password:</p> <p>Jane's password:</p>
5	<p>On Kali, download the following:</p> <p><a href="http://asecuritysite.com/files02.zip">http://asecuritysite.com/files02.zip</a></p> <p>and the files should have the following MD5 signatures:</p> <pre>MD5(1.txt)= 5d41402abc4b2a76b9719d911017c592 MD5(2.txt)= 69faab6268350295550de7d587bc323d MD5(3.txt)= fea0f1f6fede90bd0a925b4194deac11 MD5(4.txt)= d89b56f81cd7b82856231e662429bcf2</pre>	<p>Which file(s) have been modified:</p>

<b>6</b>	From your Kali, download the following ZIP file:  <a href="http://asecuritysite.com/letters.zip">http://asecuritysite.com/letters.zip</a>	View the letters. Are they different? Now determine the MD5 signature for them. What can you observe from the result?
----------	---	--

## 6 Hashing Cracking (MD5)

<http://youtu.be/Xvbk2nSzEPk>

No	Description	Result
<b>1</b>	<p>On Kali, next create a words file (<b>words</b>) with the words of “napier”, “password” “Ankle123” and “inkwell”</p> <p>Using hashcat crack the following MD5 signatures (hash1):  232DD5D7274E0D662F36C575A3BD634C  5F4DCC3B5AA765D61D8327DEB882CF99  6D5875265D1979BDAD1C8A8F383C5FF5  04013F78ACCFEC9B673005FC6F20698D</p> <p>Command used: <code>hashcat -m 0 hash1 words</code></p>	232DD...634C Is it [napier][password][Ankle123][inkwell]? 5F4DC...CF99 Is it [napier][password][Ankle123][inkwell]? 6D587...5FF5 Is it [napier][password][Ankle123][inkwell]? 04013...698D Is it [napier][password][Ankle123][inkwell]?
<b>2</b>	<p>Using the method used in the first part of this tutorial, find crack the following for names of fruits (the fruits are all in lowercase):</p> <p>FE01D67A002DFA0F3AC084298142ECCD  1F3870BE274F6C49B3E31A0C6728957F  72B302BF297A228A75730123EFEF7C41  8893DC16B1B2534BAB7B03727145A2BB  889560D93572D538078CE1578567B91A</p>	FE01D: 1F387: 72B30: 8893D: 88956:



## 7 Hashing Cracking (LM Hash/Windows)


All of the passwords in this section are in lowercase. <http://youtu.be/Xvbk2nSzEPk>

No	Description	Result
1	On Kali, and using John the Ripper, and using a word list with the names of fruits, crack the following pwdump passwords:  fred:500:E79E56A8E5C6F8FEAAD3B435B51404EE:5EBE7DFA074DA8EE8AEF1FAA2BBDE876::: bert:501:10EAF413723CBB15AAD3B435B51404EE:CA8E025E9893E8CE3D2CBF847FC56814:::	Fred: Bert:
2	On Kali, and using John the Ripper, the following pwdump passwords (they are names of major Scottish cities/towns):  Admin:500:629E2BA1C0338CE0AAD3B435B51404EE:9408CB400B20ABA3DFEC054D2B6EE5A1::: fred:501:33E58ABB4D723E5EE72C57EF50F76A05:4DFC4E7AA65D71FD4E06D061871C05F2::: bert:502:BC2B6A869601E4D9AAD3B435B51404EE:2D8947D98F0B09A88DC9FCD6E546A711:::	Admin: Fred: Bert:
3	On Kali, and using John the Ripper, crack the following pwdump passwords (they are the names of animals):  fred:500:5A8BB08EFF0D416AAAD3B435B51404EE:85A2ED1CA59D0479B1E3406972AB1928::: bert:501:C6E4266FEBEBD6A8AAD3B435B51404EE:0B9957E8BED733E0350C703AC1CDA822::: admin:502:333CB006680FAF0A417EAF50CFAC29C3:D2EDBC29463C40E76297119421D2A707:::	Fred: Bert: Admin:

Repeat all 7.1, 7.2 and 7.3 using **Ophcrack**, and the rainbow table contained on the instance (rainbow\_tables\_xp\_free).

## 8 Python tutorial

In this lab we will encrypt a string with a public key, and the decrypt with the private key.

 **Web link (Cipher code):** <https://asecuritysite.com/encryption/rsa12>

The code should be:

```
from Crypto.Util.number import *
from Crypto import Random
import Crypto
import gmpy2
import sys

bits=60
msg="Hello"

p = Crypto.Util.number.getPrime(bits, randfunc=Crypto.Random.get_random_bytes)
q = Crypto.Util.number.getPrime(bits, randfunc=Crypto.Random.get_random_bytes)

n = p*q
PHI=(p-1)*(q-1)

e=65537
d=(gmpy2.invert(e, PHI))

m= bytes_to_long(msg.encode('utf-8'))

c=pow(m,e, n)
res=pow(c,d ,n)

print "Message=%s\np=%s\nq=%s\nn=%s\ncipher=%s\ndecipher=%s" % (msg,p,q,n,c,(long_to_bytes(res)))
```

Prove the operation of the code. Now, try with 128-bit prime numbers and 256-bit prime numbers. What can you observe from the increase in the prime number size?

Can you integrate a timer in your code, so that you can assess the time to encrypt and decrypt? Now complete the following table:

Prime number size	Time to generate primes	Time to encrypt	Time to decrypt
60			
128			
256			

We can write a Python program to implement this key exchange. Enter and run the following program:

```
import random
import base64
import hashlib
```

```

import sys

g=11
p=1001

x=random.randint(5, 10)
y=random.randint(10,20)

A=(g**x) % p
B=(g**y) % p

print 'g: ',g,' (a shared value), n: ',p, ' (a prime number)'

print '\nAlice calculates:'
print 'a (Alice random): ',x
print 'Alice value (A): ',A,' (g^a) mod p'

print '\nBob calculates:'
print 'b (Bob random): ',y
print 'Bob value (B): ',B,' (g^b) mod p'

print '\nAlice calculates:'
keyA=(B**x) % p
print 'key: ',keyA,' (B^a) mod p'
print 'key: ',hashlib.sha256(str(keyA)).hexdigest()

print '\nBob calculates:'
keyB=(A**y) % p
print 'key: ',keyB,' (A^b) mod p'
print 'key: ',hashlib.sha256(str(keyB)).hexdigest()

```

Pick three different values for g and p, and make sure that the Diffie Hellman key exchange works:

g=     p=  
g=     p=  
g=     p=

Can you pick a value of g and p which will not work?

The code given below allows you to pick a value of g which will always work for a given value of p. Can you integrate the code and prove that it works?

<https://asecuritysite.com/encryption/pickg>

```
def getG(p):  
    for x in range (1,p):  
        rand = x  
        exp=1  
        next = rand % p  
  
        while (next <> 1 ):  
            next = (next*rand) % p  
            exp = exp+1  
  
        if (exp==p-1):  
            print rand  
  
print getG(p)
```

Using the prime number generates given in the RSA code, can you implement a Diffie-Hellman method which uses 256 bit prime numbers?