Lab 5: Key Exchange

Objective: Key exchange allows us to pass a shared secret key between Bob and Alice. The main methods for doing this are either encrypting with the public key, the Diffie Hellman Method and the Elliptic Curve Diffie Hellman (ECDH) method. This lab investigates these methods.

📖 **Web link (Weekly activities):** https://asecuritysite.com/esecurity/unit05

A   Diffie-Hellman

No	Description	Result
A.1	Bob and Alice have agreed on the values: g=2879, N= 9929 Bob Select x=6, Alice selects y=9	Now calculate (using a calculator): Bob's A value (g^x mod N): Alice's B value (g^Y mod N):
A.2	Now they exchange the values. Next calculate the shared key:	Bob's value (B^x mod N): Alice's value (A^Y mod N): Do they match? [Yes] [No]
A.3	If you are in the lab, select someone to share a value with. Next agree on two numbers (g and N). 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. Next exchange values.	Numbers for g and N: Your x value: Your A value: The B value you received: Shared key: Do they match: [Yes] [No]

B   OpenSSL (Diffie-Hellman and EC)

No	Description	Result
B.1	Generate 768-bit Diffie-Hellman parameters:	What is the value of g:

```openssl dhparam -out dhparams.pem 768 -text```	How many bits does the prime number have? How long does it take to produce the parameters for 1,024 bits (Group 2)? How long does it take to produce the parameters for 1536 bits (Group 5)? How would we change the g value?

No	Description	Result

B.2	Generate 768-bit Diffie-Hellman parameters:
	
	```openssl ecparam -list_curves```
	
	We can create our elliptic parameter file with:
	
	```openssl ecparam -name secp256k1 -out secp256k1.pem```
	
	Now view the details with:
	
	```openssl ecparam -in secp256k1.pem -text -param_enc explicit -noout```
	
	What are the details of the key?
	
	Now we can create our key pair:
	
	```openssl ecparam -in secp256k1.pem -genkey -noout - out mykey.pem```
		Name three 160-bit curves:
		
		By doing a search on the Internet, which curve does Bitcoin use?
		
		Can you find other application around that use this curve or others?
		
		Can you explain how you would use these EC parameters to perform the ECDH key exchange?

C   Discrete Logarithms

C.1 ElGamal and Diffie Hellman use discrete logarithms. This involves a generator value (*g*) and a prime number. A basic operation is g^x (mod p). If p=11, and g=2, determine the results (the first two have already been completed):

x	g=2, p=11 g^x (mod p)
1	2
2	8
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Note: In Python you can implement this as:

```
g=2
p=11
x=3
print g**x % p
```

What happens to the values once we go past 10?

What happens to this sequence if we use *g*=3?

C.2 We can determine the values of g which will work for a given prime number with the following:

```
import sys
import random

p=11

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)
```

Run the program and determine the possible g values for these prime numbers:

p=11: p=41: On the Internet, find a large prime number, and determine the values of g that are possible:

C.3 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=9
p=1001

a=random.randint(5, 10)

b=random.randint(10,20)

A = (g**a) % p
B = (g**b) % p

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

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

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

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

print '\nBob calculates:'
keyB=(A**b) % 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 following program sets up a man-in-the-middle attack for Eve:

```
import random
import base64
import hashlib
import sys

g=15
p=1011

a= 5
b = 9
eve = 7

message=21

A=(g**a) % p

B=(g**b) % p

Eve1 = (A**eve) % p
Eve2 = (B**eve) % p

Key1= (Eve1**a) % p
Key2= (Eve2**b) % p

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

print '\n== Random value generation ==='

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


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

print '\n==Alice sends value to Eve ==='

print 'Eve takes Alice\'s value and calculates: ',Eve1
print 'Alice gets Eve\'s value and calculates key of: ',Key1

print '\n==Bob sends value to Eve ==='

print 'Eve takes Bob\'s value and calculates: ',Eve2
print 'Bob gets Eve\'s value and calculates key of: ',Key2
```

D   Elliptic Curve Diffie-Hellman (ECDH)

ECDH is now one of the most used key exchange methods, and uses the Diffie Hellman method, but adds in elliptic curve methods. With this Alice generates (a) and Bob generates (b). We select a point on a curve (G), and Alice generates aG, and Bob generates bG. They pass the values to each other, and then Alice received bG, and Bob receives aG. Alice multiplies by a, to get abG, and Bob will multiply by b, and also get abG. This will be their shared key.

D.1   Copy and paste the code from:

📖 **Web link (ECDH):** https://asecuritysite.com/encryption/ecdh2

and confirm that Bob and Alice will always get the same shared key.

```
from os import urandom
from eccsnacks.curve25519 import scalarmult, scalarmult_base
import binascii

a = urandom(32)
a_pub = scalarmult_base(a)

b = urandom(32)
b_pub = scalarmult_base(b)

k_ab = scalarmult(a, b_pub)
k_ba = scalarmult(b, a_pub)

print "Bob public: ",binascii.hexlify(b_pub)
print "Alice public: ",binascii.hexlify(a_pub)
print "Bob shared: ",binascii.hexlify(k_ba)
print "Alice shared: ",binascii.hexlify(k_ab)
```

Do Bob and Alice end up with the same key?

How large are the random numbers that Bob and Alice generate?

Do you think that this program will be secure? How might Eve discover the shared secret?

Estimate the time it would take her to discover the key if she can try one billion keys per second:

How would you modify that program so that it was more secure?

E Simple Key Distribution Centre (KDC)

Rather than using key exchange, we can setup a KDC, and where Bob and Alice can have long-term keys. These can be used to generate a session key for them to use. Enter the following Python program, and prove its operation:

```
import hashlib
import sys
import binascii
import Padding
import random

from Crypto.Cipher import AES
from Crypto import Random

msg="test"

def encrypt(word,key, mode):
        plaintext=pad(word)
        encobj = AES.new(key,mode)
        return(encobj.encrypt(plaintext))

def decrypt(ciphertext,key, mode):
        encobj = AES.new(key,mode)
        rtn = encobj.decrypt(ciphertext)
        return(rtn)

def pad(s):
         return s
        extra = len(s) % 16
        if extra > 0:
                s = s + (' ' * (16 - extra))
        return s


rnd = random.randint(1,2**128)

keyA= hashlib.md5(str(rnd)).digest()

rnd = random.randint(1,2**128)

keyB= hashlib.md5(str(rnd)).digest()

print 'Long-term Key Alice=',binascii.hexlify(keyA)
print 'Long-term Key Bob=',binascii.hexlify(keyB)

rnd = random.randint(1,2**128)
keySession= hashlib.md5(str(rnd)).hexdigest()

ya = encrypt(keySession,keyA,AES.MODE_ECB)
yb = encrypt(keySession,keyB,AES.MODE_ECB)

print "Encrypted key sent to Alice:",binascii.hexlify(ya)
print "Encrypted key sent to Bob:",binascii.hexlify(yb)

decipherA = decrypt(ya,keyA,AES.MODE_ECB)
decipherB = decrypt(yb,keyB,AES.MODE_ECB)

print "Session key:",decipherA
print "Session key:",decipherB
```

📖 **Web link (Simple KDC):** https://asecuritysite.com/encryption/kdc01

The program above uses a shared 128-bit session key (generated by MD5). Now change the program so that you generate a 256-bit session key. What are the changes made:

F   Challenge

F.1   Bob and Alice agree on a g value of 2 (a shared value), and a prime number of 101. They then use the Diffie-Hellman key exchange method to end up with a shared key of 39. Can you write a Python program which can determine the random values they have used? Outline the values and the code here:

Can you now write a generate DH key cracker for any value of g, p? Outline code and run to evaluate the perform of our code with different ranges of the prime number (p):

G   What I should have learnt from this lab?

The key things learnt:

- The basics of the Diffie Hellman method.
- The basic method used with ECDH.