


Asymmetric Key and Key Exchange

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

1 RSA Encryption

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 | | | |

2 Key exchange

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?