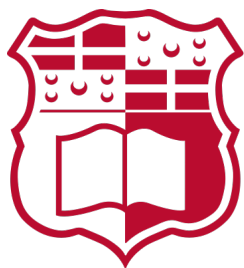


# Applied Cryptography: 2018/9 Assignment

**CPS3232**

**Hands-on Crypto**

Marc Ferriggi (286397M)



**L-Università  
ta' Malta**

**Faculty of ICT  
University of Malta  
January 2019**

# Contents

1	Block cipher modes and message authenticity . . . . .	1
2	Transparent access control on the Blockchain . . . . .	2
2.1	Implement the required reference monitor . . . . .	2
2.2	Attack 1 . . . . .	4
3	Setting up Certificate Authority (CA) trees and HTTPS servers . . . . .	6

## Block cipher modes and message authenticity

# Transparent access control on the Blockchain

## Implement the required reference monitor

The reference monitor was implemented in Jupyter Lab. The RSA modulus was generated using PyCryptodome's *RSA.generate* function, where the number of bits was set to 2048 as this was deemed a sufficient length for 2017 and the public exponent  $e$  was set to be 65537 as this is the FIPS standard. The generated key (which contains the RSA modulus  $n = p.q$ ) was then saved to a file protected by a password in order to simulate that only Rene the reference monitor has access to it.

```
%reset
from Crypto.PublicKey import RSA
import random

#Setting up FS
#Generate n:
key = RSA.generate(2048,e=65537) #nbits = 2048 (Sufficient length for
#save the secure key to a file to only be accessed by Rene
passphrase = 'Str0ngPassw0rd!%'
f = open('mykey.pem', 'wb')
f.write(key.exportKey('PEM',passphrase=passphrase)) #Use a passphras
f.close()
n = key.n #assume n is public knowledge

#Rene will then generate s, which lies in integer ring (Z_n)
s = random.randint(1,key.n+1)
#Assume this is sent to Alice and Bob over a secure channel

#Delete Secrets:
del key
del passphrase
#Authorised parties will then be given mykey.pem and with knowledge
#have access to the secure key
```

Once the RSA key was set up, Rene then set  $v$  by using the equation  $v = s^2 \bmod n$ .

```

#Rene:
#get the key
f = open('mykey.pem', 'r')
Rene_key = RSA.importKey(f.read(), passphrase='Str0ngPassw0rd!%')
f.close()
#set v
v = (s**2)%Rene_key.n

```

The interactive protocol was then implemented setting  $t = 100$ , thus having a successful attack probability of  $2^{-100}$ .

```

t = 100
for i in range(t):
    #Interactive Protocol
    #Alice
    #pick random r
    r = random.randint(1,n+1)
    x = (r**2)%n #compute x
    #x is now sent to Rene

    #Rene
    e = random.randint(0,1)
    #e is sent to Alice

    #Alice
    #if (i == 97):
    #    s = random.randint(1,n+1) #bad secret key
    y = (r*(s**e)) % n
    #y is sent to Rene

    #Rene:
    if (y**2 % n != (x*(v**e))%n):
        print('Attack!')
        break

```

### Attack 1

Weaken the implementation in a way so that Alice always commits to the same  $r \in Z_n$ . Demonstrate an attack that discloses  $s$  and which hinges on the fact that whenever  $e = 0$  Alice sends  $yr \bmod n$  as a response.

The implementation of the reference monitor was then weakened as specified. Given the fact that  $r$  wasn't changing, Oscar could simply extract  $r$  by intercepting  $y$  when  $e = 0$ . Once Oscar has  $r$  then, when  $e = 1$ , Oscar intercepts  $y$  and can extract  $s$  by using the equation  $s = y.r^{-1} \bmod n$ . The modified code to the previous implementation can be seen below.

```
from sympy import mod_inverse

gotR = 0

t = 100
#Alice will pick a random r and keep it fixed:
r = random.randint(1,n+1)
for i in range(t):
    #Interactive Protocol
    #Alice:
    x = (r**2)%n #compute x
    #x is now sent to Rene

    #Rene
    e = random.randint(0,1)
    #e is sent to Alice on open channel, Oscar can intercept this

    #Alice
    y = (r*(s**e)) % n
    #y is sent to Rene on an open channel, Oscar can intercept this

    #Oscar's Attack:
    if e==0 and gotR == 0:
        oscar_r = y
```

```

        gotR = 1
if e==1 and gotR == 1:
    oscar_s=(y*mod_inverse(r,n)) % n
    print(s==oscar_s)
    break

#Rene:
if (y**2 % n != (x*(v**e))%n):
    print( 'Attack! ')
    break

```

# **Setting up Certificate Authority (CA) trees and HTTPS servers**