Security Project

RSA

Passant Abd El Azzem Sec: 1 B.N.: 21
Reem Emad Sec: 1 B.N.: 34

22nd May 2022

—

Security Project

—

Eng. Sandra Wahid

Prof. Samir Shaheen

RSA Implementation

For all the code, we used python.
RSA Encryption(message,n, exponent):

1. Setup message(M) for encryption (Convert message to integer)
2. Public key{n,e}, e: exponent
3. Get cipher C = $M^{e}\ mod\ n$ using powMod function

RSA Decryption(c,p,q,e):

1. Private key{p,q}
2. Compute n, n=p*q
3. Compute Φ(n)=(p-1)*(q-1)
4. We know that e.d=1 mod Φ(n) so get d as the inverse of e mod Φ(n)
5. M=$C^{d}\ mod\ n$

Communication

We used socket programming.

At the sender, we initialize a host and port and then begin listening waiting for the receiver.

At the receiver, it makes the private key{p.q}, generate exponent, connect with the sender with its host and port, and first send the public key{n,e}, where n=p*q so that the sender could send the messages encrypted using the public key. The receiver waits for the sender to send messages so it can decrypt them.

Whenever the sender gets a connection, it receives the public key and then uses it to encrypt messages and send them to the receiver.

RSA Attacks

Mathematical Attack:

The attack goal is to retrieve the original plain-text message by getting the key value. It depends on factorizing the public key ‘n’. The Algorithm is as follows:

1. To know the first prime(p), try numbers from 2 until n
2. Take p whenever two conditions are satisfied:
    a. n is divisible by p
    b. p is a prime number
3. Get the second prime(q) where q=n//p
4. Decrypt the cypher the attacker has with the attacker generated private keys(p&q), to get a message
5. Encrypt this message, to get cypher text
6. Compare the attacker generated cypher with the original cypher, if they meet then those were the correct private keys, and the message in step 4 is correct.
7. If it’s not correct, repeat from step 2 until it finds the correct message or it reaches n and couldn’t know the message.

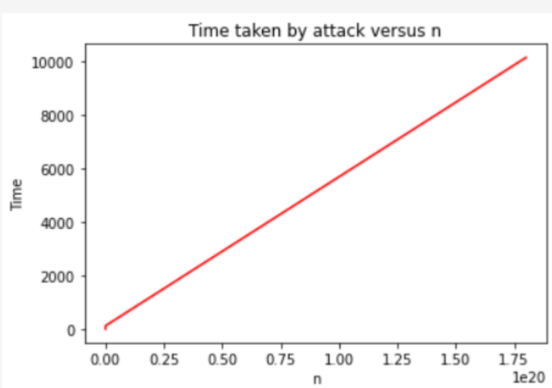
Test it by generating an array of numbers of bits and a corresponding array of messages, and plot the time taken by the attacker to know the message.

I tested it using:

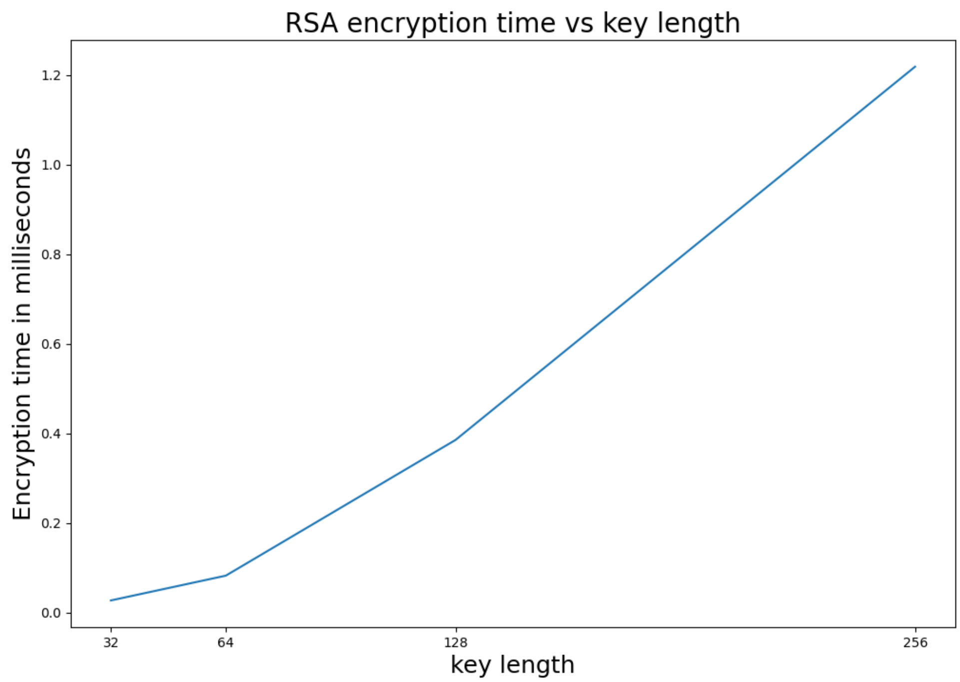
a. Array of bits = [9,12,16,17,20,24,29,34]
b. Array of messages = ["a","ab","abc",'abcd','ZZZZZ','reem12','reema12','reema123']

The results are as shown in the following graph:
It shows a great deviation as the message increases and the number of bites increases.

```
time:  8.37000000046828e-05  message:  a  p/q bits:  9  n:  175477.0
time:  0.00037380000003395253  message:  ab  p/q bits:  12  n:  4839217.0
time:  0.008173299999953088  message:  abc  p/q bits:  16  n:  1890103289.0
time:  0.03241699999989578  message:  abcd  p/q bits:  17  n:  7235768303.0
time:  0.35948309999980665  message:  ZZZZZ  p/q bits:  20  n:  686825142803.0
time:  4.127388600000131  message:  reem12  p/q bits:  24  n:  156551557298813.0
time:  120.13880989999984  message:  reema12  p/q bits:  29  n:  1.0868836705463384e+17
time:  10147.9381766  message:  reema123  p/q bits:  34  n:  1.8010815294149034e+20
```



RSA Encryption efficiency in terms of encryption time vs key length:



As expected, the time taken for encryption increases as the key length increases.
This is because of PowMod function computations due to the large value of n.

Chosen Ciphertext Attack (CCA):

The attack goal is to retrieve the original plain-text message regardless of the key value.
The Algorithm is as follows:

1. Intercept the communication between two entities.
2. Choose random integer number ‘r’
3. Encrypt ‘r’ with the RSA public key as $r^{e}\ mod\ n$
4. Multiply the encrypted r with the received ciphertext **modulo n** to get $M^{e}r^{e}\ mod\ n$
5. Send the result to the right entity.
6. The entity responds with $(M^{e}r^{e})^{d}\ mod\ n$
7. Knowing that e*d=1 mod Φ(n), this gives that the received message is actually M*r mod n
8. Divide it by the randomly generated number ‘r’ to obtain the original message.