RSA Cryptosystem Demo

A report submitted

to

MANIPAL UNIVERSITY

For Partial Fulfillment of the Requirement for the

Award of Degree

of

Bachelor of Technology

in

Computer and Communication Engineering,

by

Pratyay Amrit (140953430),
Rishabh Kanwar (140953360) and
Himank Maan (140953356)

MANIPAL
INSTITUTE OF TECHNOLOGY
A Constituent Institute of Manipal University, Manipal

October 2017

Abstract

RSA (Rivest–Shamir–Adleman) is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem".

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly.

A demonstration of the cryptosystem is implemented using python and OpenSSL, with a web interface for a better presentation.

[Cryptography]: Cryptosystems, web demo

Contents

List of Figures

1. Overall Description

1.1 Asymmetric Key Cryptography

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

- Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients:

1. Plaintext: This is the readable message or data that is fed into the algorithm as input.
2. Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.
3. Public and private keys: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
4. Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
5. Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

1.2 Essential Steps

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. Each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

2. Implementation

2.1 Basic Idea

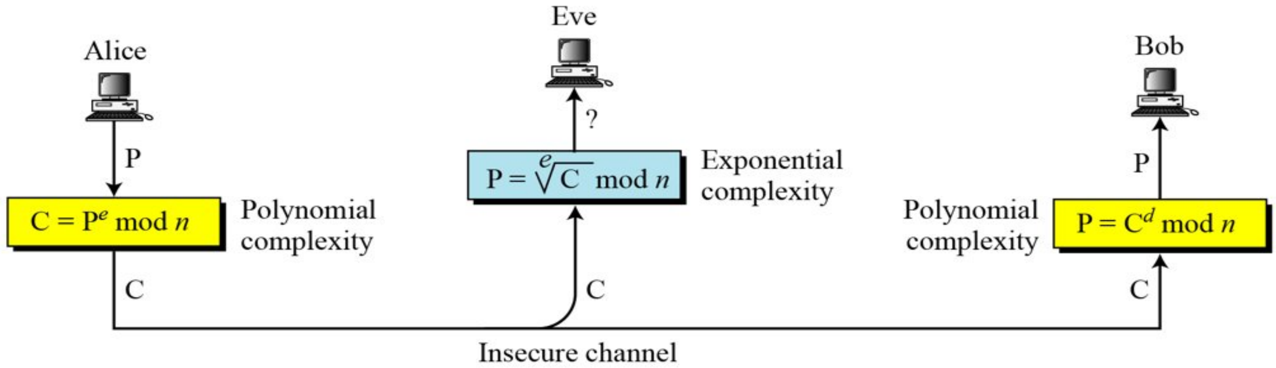

Figure 1: Basic idea of RSA

2.2 Key Generation

RSA_Key_Generation

{

Select two large primes p and q such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$

Select e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // d is inverse of e modulo $\phi(n)$

Public_key $\leftarrow$ (e, n) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

Figure 2: RSA Key Generation

- ‘p’ and ‘q’ were selected randomly using OpenSSL to generate 512 bit prime numbers.
- ‘e’ was selected as a 512 bit prime using OpenSSL, since a prime will be co-prime with every number except itself, and a 512 bit number will be smaller than a productt of two 512 bit numbers and greater than 1.
- ‘d’ was calculated using euclidean algorithm to calculate gcd, and then calculating inverse of e mod ‘n’.

2.3 Encryption

```
RSA_Encryption (P, e, n)                    // P is the plaintext in Z_n and P < n
{
   C  ←  Fast_Exponentiation (P, e, n)      // Calculation of (P^e mod n)
   return C
}
```

Figure 3: RSA Encryption

- In the implementation, the plaintext is converted into ASCII and then joined to form a number.
- For example, 'hello' => [104, 101, 108, 108, 111] => 104101108108111. The number 104101108108111 goes as plaintext in the fast exponentiation algorithm.
- Since the plaintext must be less than 'n', the maximum length for the plaintext in the implementation is limited to about 300 digits (100 characters).
- Fast exponentiation is done using the inbuilt 'pow(a, b, c)' function in python.

2.4 Decryption

```
RSA_Decryption (C, d, n)                    //C is the ciphertext in Z_n
{
   P  ←  Fast_Exponentiation (C, d, n)      // Calculation of (C^d mod n)
   return P
}
```

Figure 4: RSA Decryption

- In the implementation, fast exponentiation is applied to get the numeric plaintext
- The plaintext is then converted to characters by taking 3 digits, one at a time and converting it to it's ASCII equivalent.
- The entire string is then joined to get the plaintext.

3. Example

3.1 Key Generation

Figure 5: RSA Key Generation Demo

p =
12614793527709367465766266511753918915725732109626921165515458838336058142502769820
752151488129895885760233788092333745568707178005734362910475524006473369

q =
10128260613761537789939605121711996355254938162714131746686827075335323449300599405
248261101123586336104869319942642103457279126832400546806538825129616477

n =
12776591643743275259830032951750135837721385038313909981762170862043518668094923272
43196162308566991985131169226393837134864322156589538955066843134852437480553305970
57827137798212552716896309521864245487685861606498992115726207133777480649369878834
640235391843186774010165002873717131795101243499428284101013

fi(n) =
12776591643743275259830032951750135837721385038313909981762170862043518668094923272
43196162308566991985131169226393837134864322156589538955066843134852437253122764555
86921882092340919250981038541193973146632949404213078444344615330408254648957289581
158013526740078739034315976887412293660191526485079148011168

e =
1293254789051420261019476967983814083947852577945705819836138824961403937666585959 7
2180360816267819208841275407291096935433849716413930791241131164551 29267

d =
1108275489696339008376445762693704183397554818327910215738099369366325259855641274 6
5600379854442281086962232286048479934369032588798742620217935933277141331772079390 4
8148608574242501991492045311413930619275968710757973572806599868834640551517036604 5
31103399834451607144227968951768164348112549838945467007797 9

Public Key = (e, n); Private Key = (d)

3.2 Encryption

Figure 6: RSA Encryption Demo

plaintext = ‘hello world’

Public Key (e, n) used:

ciphertext =
1108275489696339008376445762693704183397554818327910215738099369366325259855641274 6
5600379854442281086962232286048479934369032588798742620217935933277141331772079390 4
8148608574242501991492045311413930619275968710757973572806599868834640551517036604 5
31103399834451607144227968951768164348112549838945467007797 9

3.3 Decryption

Figure 7: RSA Decryption Demo

ciphertext, private key (d) used

4. Conclusion

A web demo was implement to present how the RSA Cryptosystem works. The presentation can be used to generate keys, encrypt using public keys, or decrypt ciphertext using private keys.