TUTORIAL

CRYPTOGRAPH

Y

Asymmetric Key – Public Key Cryptography

I. RSA Encryption Algorithm

RSA encryption algorithm is a type of public-key encryption algorithm. To better understand RSA, lets first understand what is public-key encryption algorithm.

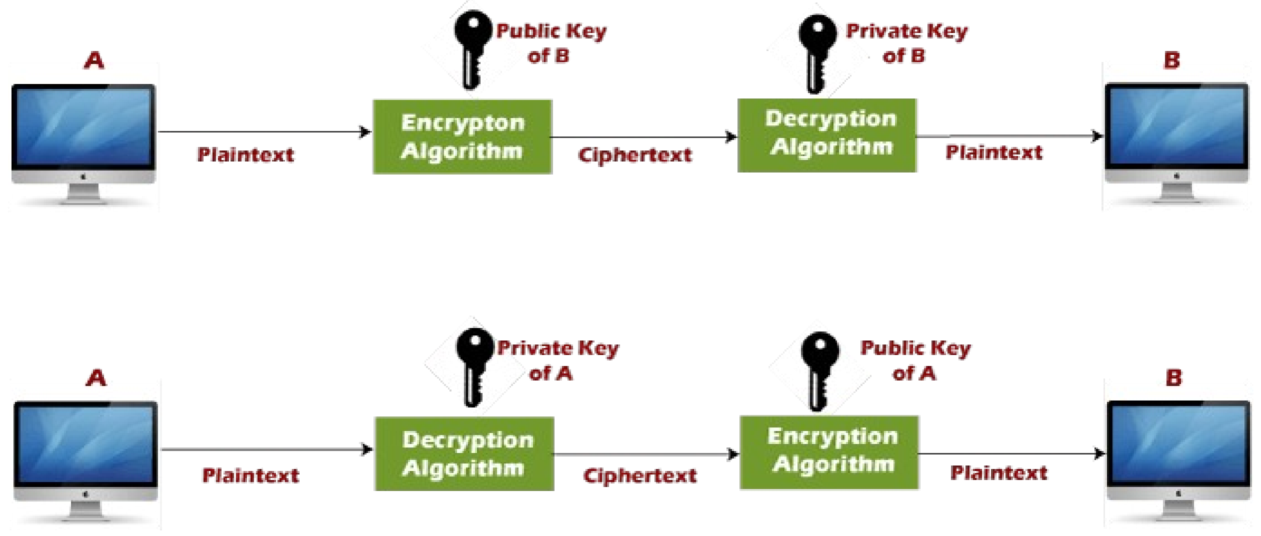
Public key encryption algorithm:

Public Key encryption algorithm is also called the Asymmetric algorithm. Asymmetric algorithms are those algorithms in which sender and receiver use different keys for encryption and decryption. Each sender is assigned a pair of keys:

- **Public key**
- **Private key**

The **Public key** is used for encryption, and the **Private Key** is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and it is known only to the user who owns the key. It means that everybody can send a message to the user using user's public key. But only the user can decrypt the message using his private key.

The Public key algorithm operates in the following manner:

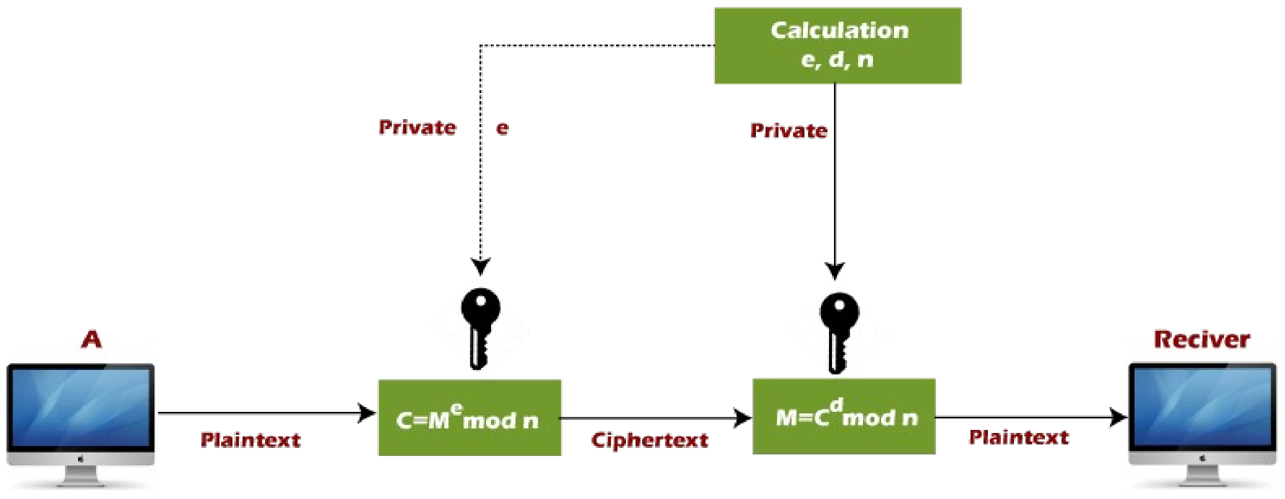


Encryption/decryption using public/private keys

- The data to be sent is encrypted by sender **A** using the public key of the intended receiver
- B decrypts the received ciphertext using its private key, which is known only to B. B replies to A encrypting its message using A's public key.
- A decrypts the received ciphertext using its private key, which is known only to him.

RSA encryption algorithm:

RSA is the most common public-key algorithm, named after its inventors **Rivest, Shamir, and Adelman (RSA).**



RSA

RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and **q**.
- Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
- Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n),

  gcd (e,d(n)) =1

- If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C = m^e \bmod n$
-

  Here**, m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  ed mod {(p - 1) x (q - 1)} = 1

  Or
  ed mod φ (n) = 1

- The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

Let's take some example of RSA encryption algorithm:
Example 1:

This example shows how we can encrypt plaintext 9 using the RSA public-key encryption algorithm. This example uses prime numbers 7 and 11 to generate the public and private keys.

Explanation:

Step 1: Select two large prime numbers, p, and **q**.

p = 7

q = 11

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 7 x 11

n = 77

Step 3: Choose a number **e** less that **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), gcd (e, d (n)) =1.

Second, we calculate

φ (n) = (p - 1) x (q-1)

φ (n) = (7 - 1) x (11 - 1)

φ (n) = 6 x 10

φ (n) = 60

Let us now choose relative prime e of 60 as 7.

Thus the public key is <e, n> = (7, 77)

Step 4: A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.

To find ciphertext from the plain text following formula is used to get ciphertext C.

$C = m^e \bmod n$

$C = 9^7 \bmod 77$

(9=1001 =>1, 10 = 2*1, 100 = 2*1, 1001 = 2*4+1; 9^1, 9^2, 9^4, 9^8)

C = 37

Step 5: The private key is <d, n>. To determine the private key, we use the following formula d such that:

de mod {(p - 1) x (q - 1)} = 1

7d mod 60 = 1, which gives d = 43.

> (60=2.2.3.5 ,
>
> 7d=d=1 mod 3 => d = 1 + 3t
>
> 7d = 2d = 1 mod 5 =>d=3 + 5r,
>
> 7d =3d =1 mod 4 => d =3+ 4k ,
>
> => d = 3 + 20w and d=1 + 3t)

The private key is <d, n> = (43, 77)

Step 6: A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.

m = c^d mod n

m = 37^{43} mod 77

m = 9

In this example, Plain text = 9 and the ciphertext = 37

Example 2:

In an RSA cryptosystem, a particular A uses two prime numbers, 13 and 17, to generate the public and private keys. If the public of A is 35. Then the private key of A is ....?.

Explanation:

Step 1: in the first step, select two large prime numbers, **p** and **q**.

p = 13

q = 17

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 13 x 17

n = 221

Step 3: Choose a number **e** less that **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), gcd (e, d (n)) =1.

Second, we calculate

φ (n) = (p - 1) x (q-1)

φ (n) = (13 - 1) x (17 - 1)

φ (n) = 12 x 16

φ (n) = 192

g.c.d (35, 192) = 1

Step 3: To determine the private key, we use the following formula to calculate the d such that:

Calculate d: de mod φ (n) = 1

d = d x 35 mod 192 = 1

d = (1 + k.φ (n))/e [let k =0, 1, 2, 3.................]

Put k = 0

d = (1 + 0 x 192)/35

d = 1/35

Put k = 1

d = (1 + 1 x 192)/35

d = 193/35

Put k = 2

d = (1 + 2 x 192)/35

d = 385/35

d = 11

The private key is <d, n> = (11, 221)

Hence, private key i.e. d = 11

Example 3:

A RSA cryptosystem uses two prime numbers 3 and 13 to generate the public key= 3 and the private key = 7. What is the value of cipher text for a plain text?

Explanation:

Step 1: In the first step, select two large prime numbers, **p** and **q**.

p = 3

q = 13

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 3 x 13

n = 39

Step 3: If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. Thus the public key is <e, n> = (3, 39).

To find ciphertext from the plain text following formula is used to get ciphertext C.

C = m^e mod n

C = 5^3 mod 39

C = 125 mod 39

C = 8

Hence, the ciphertext generated from plain text, C = 8.

Example 4:

A RSA cryptosystem uses two prime numbers, 3 and 11, to generate private key = 7. What is the value of ciphertext for a plain text 5 using the RSA public-key encryption algorithm?

Explanation:

Step 1: in the first step, select two large prime numbers, **p** and **q**.

p = 3

q = 11

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 3 x 11

n = 33

Step 3: Choose a number **e** less that **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1< e < φ (n), e is prime to φ (n), gcd (e, d (n)) =1.

Second, we calculate

φ (n) = (p - 1) x (q-1)

φ (n) = (3 - 1) x (11 - 1)

φ (n) = 2 x 10

φ (n) = 20

Step 4: To determine the public key, we use the following formula to calculate the d such that:

Calculate e x d = 1 mod φ (n)

e x 7 = 1 mod 20

e x 7 = 1 mod 20

e = (1 + k. φ (n))/ d [let k =0, 1, 2, 3……………..]

Put k = 0

e = (1 + 0 x 20) / 7

e = 1/7

Put k = 1

e = (1 + 1 x 20) / 7

e = 21/7

e = 3

The public key is <e, n> = (3, 33)

Hence, public key i.e. e = 3

https://www.javatpoint.com/rsa-encryption-algorithm

II. Practice with Public Key Cryptography in C++, C#, Java and Python:

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and the Private key is kept private.
An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Let us learn the mechanism behind the RSA algorithm:

>> Generating Public Key:

Select two prime no's. Suppose **P = 53 and Q = 59.**

Now First part of the Public key: n = P*Q = 3127.

We also need a small exponent say **e:**

But e must be an integer.

Not be a factor of n.

1 < e < Φ(n) = (P-1)(Q-1)

Let us now consider it to be equal to 3.

Our Public Key is made of n and e

>> Generating Private Key:

We need to calculate Φ(n) :

Such that **Φ(n) = (P-1)(Q-1)**

so, Φ(n) = 3016

Now calculate Private Key, **d :**

d = (k*Φ(n) + 1) / e for some integer k

For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt **"HI"**:

Convert letters to numbers: H = 8 and I = 9

Thus **Encrypted Data c = 89^e mod n.**

Thus our Encrypted Data comes out to be 1394

Now we will decrypt **1394**:

Decrypted Data = c^d mod n.

Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Below is the implementation of the RSA algorithm using different languages: C, C#, Java and Python **for** encrypting and decrypting small numeral values:

```
// C program for RSA asymmetric cryptographic
// algorithm. For demonstration values are
// relatively small compared to practical
// application
#include <bits/stdc++.h>
using namespace std;

// Returns gcd of a and b
int gcd(int a, int h)
{
    int temp;
    while (1) {
        temp = a % h;
        if (temp == 0)
            return h;
        a = h;
        h = temp;
    }
}

// Code to demonstrate RSA algorithm
int main()
{
    // Two random prime numbers
    double p = 3;
    double q = 7;

    // First part of public key:
    double n = p * q;

    // Finding other part of public key.
    // e stands for encrypt
    double e = 2;
    double phi = (p - 1) * (q - 1);
    while (e < phi) {
        // e must be co-prime to phi and
        // smaller than phi.
        if (gcd(e, phi) == 1)
            break;
        else
            e++;
    }

    // Private key (d stands for decrypt)
    // choosing d such that it satisfies
    // d*e = 1 + k * totient
    int k = 2; // A constant value
    double d = (1 + (k * phi)) / e;

    // Message to be encrypted
    double msg = 12;

    printf("Message data = %lf", msg);

    // Encryption c = (msg ^ e) % n
    double c = pow(msg, e);
    c = fmod(c, n);
    printf("\nEncrypted data = %lf", c);

    // Decryption m = (c ^ d) % n
    double m = pow(c, d);
    m = fmod(m, n);
    printf("\nOriginal Message Sent = %lf", m);

    return 0;
}
// This code is contributed by Akash Sharan.
```

Output

```
Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000
```

```
/*
 * C# program for RSA asymmetric cryptographic algorithm.
 * For demonstration, values are
 * relatively small compared to practical application
 */

using System;

public class GFG {

  public static double gcd(double a, double h)
```

```
{
  /*
       * This function returns the gcd or greatest common
       * divisor
       */
  double temp;
  while (true) {
    temp = a % h;
    if (temp == 0)
      return h;
    a = h;
    h = temp;
  }
}
static void Main()
{
  double p = 3;
  double q = 7;

  // Stores the first part of public key:
  double n = p * q;

  // Finding the other part of public key.
  // double e stands for encrypt
  double e = 2;
  double phi = (p - 1) * (q - 1);
  while (e < phi) {
    /*
          * e must be co-prime to phi and
          * smaller than phi.
          */
    if (gcd(e, phi) == 1)
      break;
    else
      e++;
  }
  int k = 2; // A constant value
  double d = (1 + (k * phi)) / e;

  // Message to be encrypted
  double msg = 12;

  Console.WriteLine("Message data = "
                    + String.Format("{0:F6}", msg));

  // Encryption c = (msg ^ e) % n
```

```
    double c = Math.Pow(msg, e);
    c = c % n;
    Console.WriteLine("Encrypted data = "
                      + String.Format("{0:F6}", c));

    // Decryption m = (c ^ d) % n
    double m = Math.Pow(c, d);
    m = m % n;
    Console.WriteLine("Original Message Sent = "
                      + String.Format("{0:F6}", m));
  }
}
// This code is contributed by Pranay Arora.
```

Output

```
Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000
```

```
/*
 * Java program for RSA asymmetric cryptographic algorithm.
 * For demonstration, values are
 * relatively small compared to practical application
 */
import java.io.*;
import java.math.*;
import java.util.*;

public class GFG {
  public static double gcd(double a, double h)
  {
    /*
         * This function returns the gcd or greatest common
         * divisor
         */
    double temp;
    while (true) {
      temp = a % h;
      if (temp == 0)
        return h;
      a = h;
      h = temp;
    }
  }
  public static void main(String[] args)
  {
    double p = 3;
    double q = 7;

    // Stores the first part of public key:
    double n = p * q;

    // Finding the other part of public key.
    // double e stands for encrypt
    double e = 2;
    double phi = (p - 1) * (q - 1);
    while (e < phi) {
      /*
             * e must be co-prime to phi and
             * smaller than phi.
             */
      if (gcd(e, phi) == 1)
        break;
      else
        e++;
    }
    int k = 2; // A constant value
    double d = (1 + (k * phi)) / e;

    // Message to be encrypted
    double msg = 12;

    System.out.println("Message data = " + msg);

    // Encryption c = (msg ^ e) % n
    double c = Math.pow(msg, e);
    c = c % n;
    System.out.println("Encrypted data = " + c);

    // Decryption m = (c ^ d) % n
    double m = Math.pow(c, d);
    m = m % n;
    System.out.println("Original Message Sent = " + m);
  }
}

// This code is contributed by Pranay Arora.
```

Output

```
Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000
```

```python
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math


def gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp


p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):

    # e must be co-prime to phi and
    # smaller than phi.
    if(gcd(e, phi) == 1):
        break
    else:
        e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient

k = 2
d = (1 + (k*phi))/e
```

```
# Message to be encrypted
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)


# This code is contributed by Pranay Arora.
```

Output

```
Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000
```

https://www.geeksforgeeks.org/rsa-algorithm-cryptography/?ref=lbp

III. Practice with Public Key Cryptography in Java with GUI: RSA

Download code here:

https://drive.google.com/drive/folders/1XwXOXOgU_VJNrUebmgipXHd18XVq8Woa?usp=share_link

(https://stackjava.com/demo/code-java-vi-du-ma-hoa-giai-ma-voi-rsa.html)

You should compile run GenerateKeys.java first to generate key pair (private key and public key) then encrypt message and decrypt cipher text using those key pair.