RSA Algorithm in Cryptography

Last Updated : 09 Nov, 2023

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 Φ(n).
1 < e < Φ(n) [Φ(n) is discussed below],
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 for

Method 1: Encrypting and decrypting small numeral values:

C++

```
// 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
```

```
        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.
```

Java

```
/*package whatever //do not write package name here */
import java.io.*;
import java.math.*;
import java.util.*;
/*
 * Java program for RSA asymmetric cryptographic algorithm.
 * For demonstration, values are
 * relatively small compared to practical application
 */
public class GFG {
    public static double gcd(double a, double h)
    {
        /*
         * This function returns the gcd or greatest common
```

```
        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;
```

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

Python3

```
# 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
```

```
# 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.
```

C#

```
/*
 * 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;
```

```
        // 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.
```

Javascript

```
//GFG
//Javascript code for this approach
function gcd(a, h) {
  /*
   * This function returns the gcd or greatest common
```

```
  while (true) {
    temp = a % h;
    if (temp == 0) return h;
    a = h;
    h = temp;
  }
}

let p = 3;
let q = 7;

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

// Finding the other part of public key.
// e stands for encrypt
let e = 2;
let 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++;
}
let k = 2; // A constant value
let d = (1 + (k * phi)) / e;

// Message to be encrypted
let msg = 12;

console.log("Message data = " + msg);

// Encryption c = (msg ^ e) % n
let c = Math.pow(msg, e);
c = c % n;
console.log("Encrypted data = " + c);

// Decryption m = (c ^ d) % n
let m = Math.pow(c, d);
m = m % n;
console.log("Original Message Sent = " + m);

//This code is written by Sundaram
```

Output

```
Message data = 12.000000
Encrypted data = 3.000000
Original Message Sent = 12.000000
```

Method 2: Encrypting and decrypting plain text messages containing alphabets and numbers using their ASCII value:

C++

```
#include <bits/stdc++.h>
using namespace std;
set<int>
    prime; // a set will be the collection of prime numbers,
           // where we can select random primes p and q
int public_key;
int private_key;
int n;
// we will run the function only once to fill the set of
// prime numbers
void primefiller()
{
    // method used to fill the primes set is seive of
    // eratosthenes(a method to collect prime numbers)
    vector<bool> seive(250, true);
    seive[0] = false;
    seive[1] = false;
    for (int i = 2; i < 250; i++) {
        for (int j = i * 2; j < 250; j += i) {
            seive[j] = false;
        }
    } // filling the prime numbers
    for (int i = 0; i < seive.size(); i++) {
        if (seive[i])
            prime.insert(i);
    }
}
```

```
{
    int k = rand() % prime.size();
    auto it = prime.begin();
    while (k--)
        it++;
    int ret = *it;
    prime.erase(it);
    return ret;
}
void setkeys()
{
    int prime1 = pickrandomprime(); // first prime number
    int prime2 = pickrandomprime(); // second prime number
    // to check the prime numbers selected
    // cout<<prime1<<" "<<prime2<<endl;
    n = prime1 * prime2;
    int fi = (prime1 - 1) * (prime2 - 1);
    int e = 2;
    while (1) {
        if (__gcd(e, fi) == 1)
            break;
        e++;
    } // d = (k*Φ(n) + 1) / e for some integer k
    public_key = e;
    int d = 2;
    while (1) {
        if ((d * e) % fi == 1)
            break;
        d++;
    }
    private_key = d;
}
// to encrypt the given number
long long int encrypt(double message)
{
    int e = public_key;
    long long int encrpyted_text = 1;
    while (e--) {
        encrpyted_text *= message;
        encrpyted_text %= n;
    }
    return encrpyted_text;
}
// to decrypt the given number
long long int decrypt(int encrpyted_text)
{
```

```
            decrypted *= encrpyted_text;
            decrypted %= n;
        }
        return decrypted;
    }
    // first converting each character to its ASCII value and
    // then encoding it then decoding the number to get the
    // ASCII and converting it to character
    vector<int> encoder(string message)
    {
        vector<int> form;
        // calling the encrypting function in encoding function
        for (auto& letter : message)
            form.push_back(encrypt((int)letter));
        return form;
    }
    string decoder(vector<int> encoded)
    {
        string s;
        // calling the decrypting function decoding function
        for (auto& num : encoded)
            s += decrypt(num);
        return s;
    }
    int main()
    {
        primefiller();
        setkeys();
        string message = "Test Message";
        // uncomment below for manual input
        // cout<<"enter the message\n";getline(cin,message);
        // calling the encoding function
        vector<int> coded = encoder(message);
        cout << "Initial message:\n" << message;
        cout << "\n\nThe encoded message(encrypted by public "
                "key)\n";
        for (auto& p : coded)
            cout << p;
        cout << "\n\nThe decoded message(decrypted by private "
                "key)\n";
        cout << decoder(coded) << endl;
        return 0;
    }
```

Java

```
import java.util.List;
import java.util.Random;

public class GFG {
    private static HashSet<Integer> prime = new HashSet<>();
    private static Integer public_key = null;
```



```
    private static Random random = new Random();

    public static void main(String[] args)
    {
        primeFiller();
        setKeys();
        String message = "Test Message";
        // Uncomment below for manual input
        // System.out.println("Enter the message:");
        // message = new Scanner(System.in).nextLine();

        List<Integer> coded = encoder(message);

        System.out.println("Initial message:");
        System.out.println(message);
        System.out.println(
            "\n\nThe encoded message (encrypted by public key)\n");
        System.out.println(
            String.join("", coded.stream()
                                 .map(Object::toString)
                                 .toArray(String[] ::new)));
        System.out.println(
            "\n\nThe decoded message (decrypted by public key)\n");
        System.out.println(decoder(coded));
    }

    public static void primeFiller()
    {
        boolean[] sieve = new boolean[250];
        for (int i = 0; i < 250; i++) {
            sieve[i] = true;
        }

        sieve[0] = false;
        sieve[1] = false;

        for (int i = 2; i < 250; i++) {
            for (int j = i * 2; j < 250; j += i) {
```

```
        for (int i = 0; i < sieve.length; i++) {
            if (sieve[i]) {
                prime.add(i);
            }
        }
    }

    public static int pickRandomPrime()
    {
        int k = random.nextInt(prime.size());
        List<Integer> primeList = new ArrayList<>(prime);
        int ret = primeList.get(k);
        prime.remove(ret);
        return ret;
    }

    public static void setKeys()
    {
        int prime1 = pickRandomPrime();
        int prime2 = pickRandomPrime();

        n = prime1 * prime2;
        int fi = (prime1 - 1) * (prime2 - 1);

        int e = 2;
        while (true) {
            if (gcd(e, fi) == 1) {
                break;
            }
            e += 1;
        }

        public_key = e;

        int d = 2;
        while (true) {
            if ((d * e) % fi == 1) {
                break;
            }
            d += 1;
        }

        private_key = d;
    }
```

```
        int encrypted_text = 1;
        while (e > 0) {
            encrypted_text *= message;
            encrypted_text %= n;
            e -= 1;
        }
        return encrypted_text;
    }

    public static int decrypt(int encrypted_text)
    {
        int d = private_key;
        int decrypted = 1;
        while (d > 0) {
            decrypted *= encrypted_text;
            decrypted %= n;
            d -= 1;
        }
        return decrypted;
    }

    public static int gcd(int a, int b)
    {
        if (b == 0) {
            return a;
        }
        return gcd(b, a % b);
    }

    public static List<Integer> encoder(String message)
    {
        List<Integer> encoded = new ArrayList<>();
        for (char letter : message.toCharArray()) {
            encoded.add(encrypt((int)letter));
        }
        return encoded;
    }

    public static String decoder(List<Integer> encoded)
    {
        StringBuilder s = new StringBuilder();
        for (int num : encoded) {
            s.append((char)decrypt(num));
        }
        return s.toString();
    }
```

Python3

```python
import random
import math

# A set will be the collection of prime numbers,
# where we can select random primes p and q
prime = set()

public_key = None
private_key = None
n = None

# We will run the function only once to fill the set of
# prime numbers
def primefiller():
    # Method used to fill the primes set is Sieve of
    # Eratosthenes (a method to collect prime numbers)
    seive = [True] * 250
    seive[0] = False
    seive[1] = False
    for i in range(2, 250):
        for j in range(i * 2, 250, i):
            seive[j] = False

    # Filling the prime numbers
    for i in range(len(seive)):
        if seive[i]:
            prime.add(i)


# Picking a random prime number and erasing that prime
# number from list because p!=q
def pickrandomprime():
    global prime
    k = random.randint(0, len(prime) - 1)
    it = iter(prime)
    for _ in range(k):
        next(it)

    ret = next(it)
    prime.remove(ret)
    return ret
```

```
    prime2 = pickrandomprime()  # Second prime number

    n = prime1 * prime2
    fi = (prime1 - 1) * (prime2 - 1)

    e = 2
    while True:
        if math.gcd(e, fi) == 1:
            break
        e += 1

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

    d = 2
    while True:
        if (d * e) % fi == 1:
            break
        d += 1

    private_key = d


# To encrypt the given number
def encrypt(message):
    global public_key, n
    e = public_key
    encrypted_text = 1
    while e > 0:
        encrypted_text *= message
        encrypted_text %= n
        e -= 1
    return encrypted_text


# To decrypt the given number
def decrypt(encrypted_text):
    global private_key, n
    d = private_key
    decrypted = 1
    while d > 0:
        decrypted *= encrypted_text
        decrypted %= n
        d -= 1
    return decrypted
```

```
# ASCII and converting it to character
def encoder(message):
    encoded = []
    # Calling the encrypting function in encoding function
    for letter in message:
        encoded.append(encrypt(ord(letter)))
    return encoded


def decoder(encoded):
    s = ''
    # Calling the decrypting function decoding function
    for num in encoded:
        s += chr(decrypt(num))
    return s


if __name__ == '__main__':
    primefiller()
    setkeys()
    message = "Test Message"
    # Uncomment below for manual input
    # message = input("Enter the message\n")
    # Calling the encoding function
    coded = encoder(message)

    print("Initial message:")
    print(message)
    print("\n\nThe encoded message(encrypted by public key)\n")
    print(''.join(str(p) for p in coded))
    print("\n\nThe decoded message(decrypted by public key)\n")
    print(''.join(str(p) for p in decoder(coded)))
```

C#

```
using System;
using System.Collections.Generic;

public class GFG
{
    private static HashSet<int> prime = new HashSet<int>();
    private static int? public_key = null;
    private static int? private_key = null;
```

```
public static void Main()
{
    PrimeFiller();
    SetKeys();
    string message = "Test Message";
    // Uncomment below for manual input
    // Console.WriteLine("Enter the message:");
    // message = Console.ReadLine();

    List<int> coded = Encoder(message);

    Console.WriteLine("Initial message:");
    Console.WriteLine(message);
    Console.WriteLine("\n\nThe encoded message (encrypted by public key)\n");
    Console.WriteLine(string.Join("", coded));
    Console.WriteLine("\n\nThe decoded message (decrypted by public key)\n");
    Console.WriteLine(Decoder(coded));
}

public static void PrimeFiller()
{
    bool[] sieve = new bool[250];
    for (int i = 0; i < 250; i++)
    {
        sieve[i] = true;
    }

    sieve[0] = false;
    sieve[1] = false;

    for (int i = 2; i < 250; i++)
    {
        for (int j = i * 2; j < 250; j += i)
        {
            sieve[j] = false;
        }
    }

    for (int i = 0; i < sieve.Length; i++)
    {
        if (sieve[i])
        {
            prime.Add(i);
        }
    }
}
```

```
        int k = random.Next(0, prime.Count - 1);
        var enumerator = prime.GetEnumerator();
        for (int i = 0; i <= k; i++)
        {
            enumerator.MoveNext();
        }

        int ret = enumerator.Current;
        prime.Remove(ret);
        return ret;
    }

    public static void SetKeys()
    {
        int prime1 = PickRandomPrime();
        int prime2 = PickRandomPrime();

        n = prime1 * prime2;
        int fi = (prime1 - 1) * (prime2 - 1);

        int e = 2;
        while (true)
        {
            if (GCD(e, fi) == 1)
            {
                break;
            }
            e += 1;
        }

        public_key = e;

        int d = 2;
        while (true)
        {
            if ((d * e) % fi == 1)
            {
                break;
            }
            d += 1;
        }

        private_key = d;
    }
     public static int Encrypt(int message)
    {
```

```
        {
            encrypted_text *= message;
            encrypted_text %= n.Value;
            e -= 1;
        }
        return encrypted_text;
    }

    public static int Decrypt(int encrypted_text)
    {
        int d = private_key.Value;
        int decrypted = 1;
        while (d > 0)
        {
            decrypted *= encrypted_text;
            decrypted %= n.Value;
            d -= 1;
        }
        return decrypted;
    }
    public static int GCD(int a, int b)
    {
        if (b == 0)
        {
            return a;
        }
        return GCD(b, a % b);
    }

    public static List<int> Encoder(string message)
    {
        List<int> encoded = new List<int>();
        foreach (char letter in message)
        {
            encoded.Add(Encrypt((int)letter));
        }
        return encoded;
    }

    public static string Decoder(List<int> encoded)
    {
        string s = "";
        foreach (int num in encoded)
        {
            s += (char)Decrypt(num);
        }
```

```
}
```

Output

```
Initial message:
Test Message

The encoded message(encrypted by public key)
863312887135951593413927434912887135951359583051879012887

The decoded message(decrypted by private key)
Test Message
```

Implementation of RSA Cryptosystem using Primitive Roots in C++

we will implement a simple version of RSA using primitive roots.

Step 1: Generating Keys

To start, we need to generate two large prime numbers, p and q. These primes should be of roughly equal length and their product should be much larger than the message we want to encrypt.

We can generate the primes using any primality testing algorithm, such as the Miller-Rabin test. Once we have the two primes, we can compute their product n = p*q, which will be the modulus for our RSA system.

Next, we need to choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1, where phi(n) = (p-1)*(q-1) is Euler's totient function. This value of e will be

To compute the private key exponent d, we need to find an integer d such that d*e = 1 (mod phi(n)). This can be done using the extended Euclidean algorithm.

Our public key is (n, e) and our private key is (n, d).

Step 2: Encryption

To encrypt a message m, we need to convert it to an integer between 0 and n-1. This can be done using a reversible encoding scheme, such as ASCII or UTF-8.

Once we have the integer representation of the message, we compute the ciphertext c as c = m^e (mod n). This can be done efficiently using modular exponentiation algorithms, such as binary exponentiation.

Step 3: Decryption

To decrypt the ciphertext c, we compute the plaintext m as m = c^d (mod n). Again, we can use modular exponentiation algorithms to do this efficiently.

Step 4: Example

Let's walk through an example using small values to illustrate how the RSA cryptosystem works.

Suppose we choose p = 11 and q = 13, giving us n = 143 and phi(n) = 120. We can choose e = 7, since gcd(7, 120) = 1. Using the extended Euclidean algorithm, we can compute d = 103, since 7*103 = 1 (mod 120).

Our public key is (143, 7) and our private key is (143, 103).

Suppose we want to encrypt the message "HELLO". We can convert this to the integer 726564766, using ASCII encoding. Using the public key, we compute the ciphertext as c = 726564766^7 (mod 143) = 32.

To decrypt the ciphertext, we use the private key to compute m = 32^103 (mod 143) = 726564766, which is the original message.

Example Code:

C++

```
using namespace std;

// calculate phi(n) for a given number n
int phi(int n) {
    int result = n;
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            result -= result / i;
        }
    }
    if (n > 1) {
        result -= result / n;
    }
    return result;
}

// calculate gcd(a, b) using the Euclidean algorithm
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

// calculate a^b mod m using modular exponentiation
int modpow(int a, int b, int m) {
    int result = 1;
    while (b > 0) {
        if (b & 1) {
            result = (result * a) % m;
        }
        a = (a * a) % m;
        b >>= 1;
    }
    return result;
}

// generate a random primitive root modulo n
int generatePrimitiveRoot(int n) {
    int phiN = phi(n);
    int factors[phiN], numFactors = 0;
    int temp = phiN;
    // get all prime factors of phi(n)
```

```
            while (temp % i == 0) {
                temp /= i;
            }
        }
    }
    if (temp > 1) {
        factors[numFactors++] = temp;
    }
    // test possible primitive roots
    for (int i = 2; i <= n; i++) {
        bool isRoot = true;
        for (int j = 0; j < numFactors; j++) {
            if (modpow(i, phiN / factors[j], n) == 1) {
                isRoot = false;
                break;
            }
        }
        if (isRoot) {
            return i;
        }
    }
    return -1;
}

int main() {
    int p = 61;
    int q = 53;
    int n = p * q;
    int phiN = (p - 1) * (q - 1);
    int e = generatePrimitiveRoot(phiN);
    int d = 0;
    while ((d * e) % phiN != 1) {
        d++;
    }
    cout << "Public key: {" << e << ", " << n << "}" << endl;
    cout << "Private key: {" << d << ", " << n << "}" << endl;
    int m = 123456;
    int c = modpow(m, e, n);
    int decrypted = modpow(c, d, n);
    cout << "Original message: " << m << endl;
    cout << "Encrypted message: " << c << endl;
    cout << "Decrypted message: " << decrypted << endl;
    return 0;
}
```

```
Public key: {3, 3233}
Private key: {2011, 3233}
Original message: 123456
Encrypted message: 855
Decrypted message: 123456
```

Advantages:

- **Security:** RSA algorithm is considered to be very secure and is widely used for secure data transmission.
- **Public-key cryptography:** RSA algorithm is a public-key cryptography algorithm, which means that it uses two different keys for encryption and decryption. The public key is used to encrypt the data, while the private key is used to decrypt the data.
- **Key exchange:** RSA algorithm can be used for secure key exchange, which means that two parties can exchange a secret key without actually sending the key over the network.
- **Digital signatures:** RSA algorithm can be used for digital signatures, which means that a sender can sign a message using their private key, and the receiver can verify the signature using the sender's public key.
- **Speed:** The RSA technique is suited for usage in real-time applications since it is quite quick and effective.
- **Widely used:** Online banking, e-commerce, and secure communications are just a few fields and applications where the RSA algorithm is extensively developed.

Disadvantages:

- **Slow processing speed:** RSA algorithm is slower than other encryption algorithms, especially when dealing with large amounts of data.
- **Large key size:** RSA algorithm requires large key sizes to be secure, which means that it requires more computational resources and storage space.
- **Vulnerability to side-channel attacks:** RSA algorithm is vulnerable to side-

through side channels such as power consumption, electromagnetic radiation, and timing analysis to extract the private key.

- **Limited use in some applications:** RSA algorithm is not suitable for some applications, such as those that require constant encryption and decryption of large amounts of data, due to its slow processing speed.
- **Complexity:** The RSA algorithm is a sophisticated mathematical technique that some individuals may find challenging to comprehend and use.
- **Key Management:** The secure administration of the private key is necessary for the RSA algorithm, although in some cases this can be difficult.
- **Vulnerability to Quantum Computing:** Quantum computers have the ability to attack the RSA algorithm, potentially decrypting the data.

Previous
Difference between AES and DES ciphers

Next
Implementation of Diffie-Hellman Algorithm

Share your thoughts in the comments

Add Your Comment

Similar Reads

- Custom Building Cryptography Algorithms (Hybrid Cryptography)
- Classical Cryptography and Quantum Cryptography
- RSA Algorithm using Multiple Precision Arithmetic Library
- How to generate Large Prime numbers for RSA Algorithm
- Difference between RSA algorithm and DSA
- How to solve RSA Algorithm Problems?
- Shamir's Secret Sharing Algorithm | Cryptography
- Knapsack Encryption Algorithm in Cryptography
- CAST Algorithm in Cryptography
- Weak RSA decryption with Chinese-remainder theorem

GeeksforGeeks

Article Tags : cryptography , number-theory , Computer Networks

Practice Tags : number-theory

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305