1. **INTRODUCTION TO CRYPTOGRAPHY:**

**Cryptography** is the science and art of designing and using codes to secure information and communications. The word cryptography comes from the Greek words kryptos, meaning hidden, and graphein, meaning writing. Cryptography has a long and fascinating history, dating back to ancient times, when people used simple substitution and transposition ciphers to conceal their messages. Today, cryptography has evolved into a complex and sophisticated field that relies on mathematics, computer science, engineering and physics.

The *main goal of cryptography* is to ensure the confidentiality, integrity, authenticity and availability of data and users. Confidentiality means that only the intended recipients can access the data. Integrity means that the data is not altered or corrupted during transmission or storage. Authenticity means that the data comes from a verified source and has not been forged or impersonated. Availability means that the data is accessible and usable when needed.

To achieve these goals, cryptography uses two main processes: encryption and decryption. Encryption is the process of transforming plain text (the original data) into cipher text (the coded data) using a key (a secret parameter). Decryption is the reverse process of recovering plain text from cipher text using the same or a different key. There are different types of encryption and decryption methods, such as symmetric key, asymmetric key and hash functions.

Cryptography has many applications in various domains, such as computer security, digital currencies, web browsing, authentication and cryptocurrencies. Cryptography helps to protect data and users from unauthorized access, tampering, theft and fraud. Cryptography also enables new forms of communication, collaboration and commerce that are based on trust, privacy and security.

*In this document*, we will explore the concepts, types, examples and challenges of cryptography in more detail considering the cryptography techniques like: **RSA**(Rivest, Shamir, Adleman), We will also discuss the performance analysis of these cryptographic techniques and compare these techniques with each other to ensure which will be more useful depending upon our need and requirement.

1. **NEED FOR CRYPTOGRAPHY:**

Some applications of cryptography are described below-

* End-to-end encryption: A type of cryptography that encrypts messages with different keys for the sender and the receiver. No one else can read the messages, even if they intercept them. Used by apps like WhatsApp, Signal and Telegram.
* Authentication: A type of cryptography that checks the identity of a user or a device before allowing access. It encrypts or hashes credentials, such as passwords, tokens or biometrics. Used by systems and services like email, online banking and VPN.
* Electronic signatures: A type of cryptography that lets users sign documents digitally. It uses different keys to generate and verify signatures. The signature proves who signed the document and if it was changed. Used by apps like DocuSign, Adobe Sign and HelloSign.
* Secure web browsing: A type of cryptography that protects web traffic from being spied on or altered. It uses TLS or SSL protocols, which encrypt data with the same or different keys. The protocol also checks the identity of the web server and the web browser. Used by websites like Google, Facebook and Amazon.
* Computer passwords: A type of cryptography that secures access to computer accounts or devices. It uses hash functions, which produce a unique output from any input. The password is hashed before storing or sending it, so it cannot be read or cracked. Used by systems and devices like Windows, macOS and Android.

1. **PRINCIPLES OF CRYPTOGRAPHY:**

The Building Principle of cryptography are described below:

* *Confidentiality*:

In the realm of cryptography, confidentiality is paramount. It ensures that information remains secret and accessible only to the intended sender and receiver. Any compromise in confidentiality occurs if unauthorized entities manage to intercept and access the encrypted message. For example, if sender A encrypts confidential information for receiver B, an attacker C intercepting and accessing this information constitutes a breach of confidentiality.

* *Authentication*:

Cryptographic authentication verifies the identity of users, systems, or entities involved in communication. This process often employs mechanisms like usernames and passwords to ensure that only authorized individuals with pre-registered identities can access sensitive information securely.

* *Integrity:*

The principle of integrity guarantees that the information received is accurate and unchanged during transmission. In cryptography, both system integrity and data integrity are crucial. System integrity ensures that a cryptographic system functions as intended without unauthorized manipulation, while data integrity ensures that information remains unaltered in storage and during transmission.

* *Non-Repudiation*:

Non-repudiation mechanisms in cryptography prevent the denial of sent messages. It ensures that a sender cannot later deny sending a message, providing a level of assurance and accountability in cryptographic communication.

* *Access Control:*

Access control in cryptography involves role management and rule management. Role management dictates who should have access to cryptographic data, while rule management determines the extent of that access. The displayed information is contingent on the access privileges of the individual accessing it.

* *Availability* :

The principle of availability in cryptography emphasizes that cryptographic resources should be consistently accessible to authorized parties. For cryptographic systems, ensuring information availability is crucial to fulfilling user requests effectively.

* *Ethical and Legal Issues* :

Ethical considerations in cryptography revolve around individuals' right to privacy, property concerns related to information ownership, organizational rights to collect information (accessibility), and the obligation to maintain information accuracy, authenticity, and fidelity within cryptographic practices. Adhering to ethical principles is essential in navigating the complexities of cryptographic systems in alignment with legal frameworks.

1. **Types of Cryptography:**

On the basis of encryption techniques on the data, the cryptographic techniques are divided in two parts-

* *Symmetric key cryptography*:

It is an encryption technique that uses one key to encrypt and decrypt messages. The key is shared by the sender and the receiver, and it must be kept secret from anyone else who might want to access the messages. Symmetric key cryptography is also called *secret key cryptography* or *private key cryptography* , because only the parties who know the secret key can communicate securely. Symmetric key cryptography is commonly used in banking and data storage applications to prevent fraud, identity theft and data breaches. These applications require fast and efficient encryption and decryption of large amounts of data, which symmetric key cryptography can provide. Some examples of symmetric key algorithms are AES, DES, RC4 and Blowfish. These algorithms use different methods to transform the plain text into cipher text and vice versa, using the same key.

* *Asymmetric key cryptography* :

it is also known as public-key cryptography, represents a revolutionary paradigm in securing digital communication. Unlike traditional symmetric key cryptography, which relies on a shared secret key between communicating parties, asymmetric key cryptography employs a pair of distinct but mathematically linked keys – a public key and a private key. The public key is openly shared and used for encryption, while the private key, known only to the key's owner, is employed for decryption. This duality enables secure and efficient communication across untrusted networks. The strength of asymmetric key cryptography lies in its ability to provide a secure method for key exchange, authentication, and digital signatures without necessitating a priori sharing of secret keys. This cryptographic approach underpins the security infrastructure of various technologies, including secure web browsing (HTTPS), digital signatures, and email encryption, ensuring the confidentiality and integrity of digital communications in an increasingly interconnected and data-centric world.

1. **RSA ALGORITHM:**

*What Is the RSA Algorithm*?

The RSA algorithm is a public-key signature algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adleman. Their paper was first published in 1977, and the algorithm uses logarithmic functions to keep the working complex enough to withstand brute force and streamlined enough to be fast post-deployment. The image below shows it verifies the digital signatures using RSA methodology.

RSA can also encrypt and decrypt general information to securely exchange data along with handling digital signature verification. The image above shows the entire procedure of the RSA algorithm. We will understand more about it in the next section.[1]

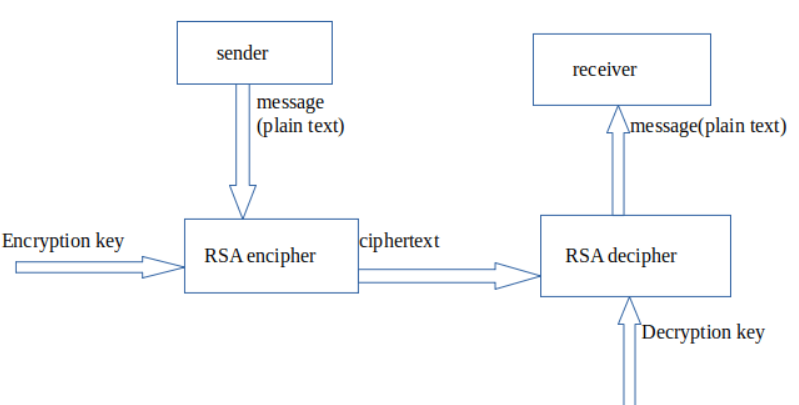


Fig- ii.a

1. **RSA in Data Encryption:**

When using RSA for encryption and decryption of general data, it reverses the key set usage. Unlike signature verification, it uses the receiver’s public key to encrypt the data, and it uses the receiver’s private key in decrypting the data. Thus, there is no need to exchange any keys in this scenario.

There are two broad components when it comes to RSA cryptography, they are:

1. Key Generation:

Generating the keys to be used for encrypting and decrypting the data to be exchanged.

1. Encryption/Decryption Function:

The steps that need to be run when scrambling and recovering the data.

We will now understand each of these steps in our next sub-topic.[1]

1. ***Steps in RSA Algorithm:***

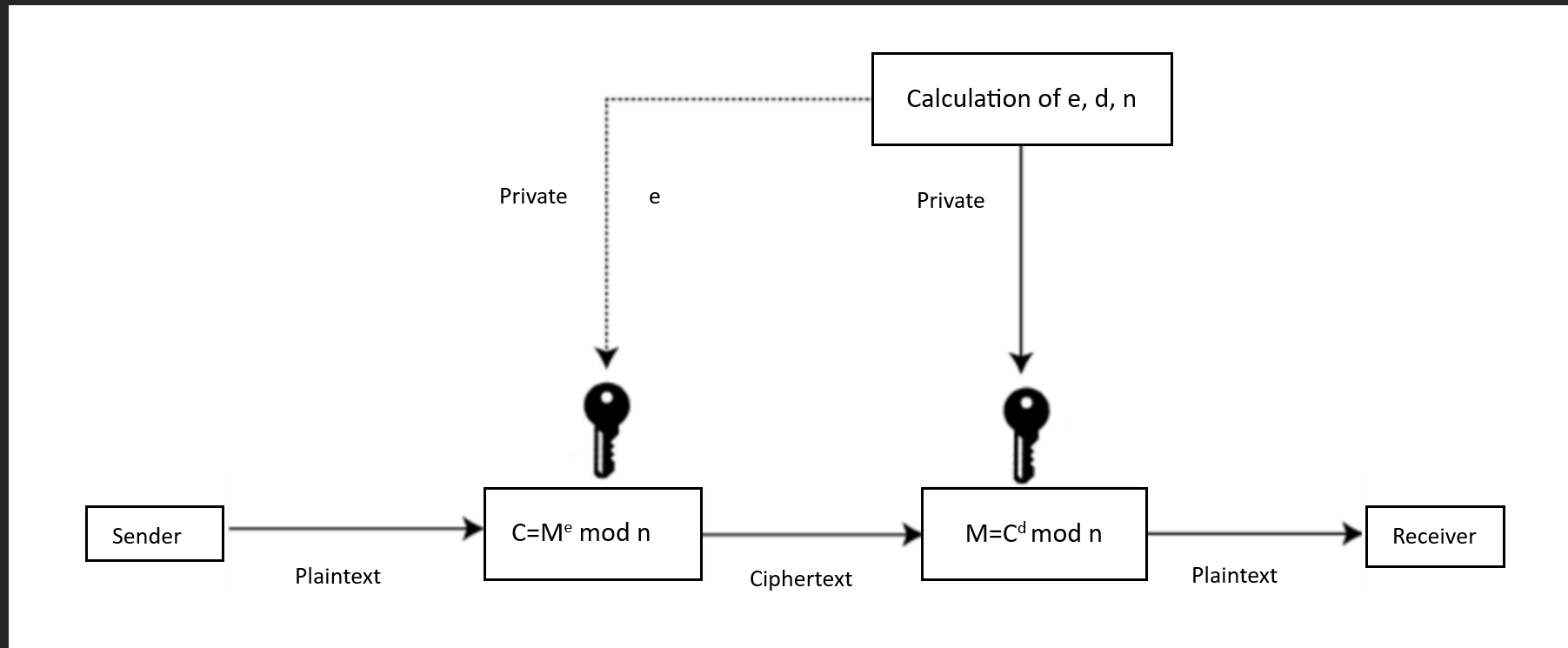


Fig- ii.b

Consider the [Fig- ii.b] to understand the below steps

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

1. Select two large prime numbers, p and **q**.
2. Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
3. 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**
4. 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, the following formula is used to get ciphertext C.  
   **C = me mod 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.
5. To determine the private key, we use the following formula to calculate the d such that:  
   **De mod {(p - 1) x (q - 1)} = 1  
   Or  
   De mod φ (n) = 1**
6. The private key is <d, n>. A ciphertext message **c** is decrypted using the private key <d, n>. To calculate plain text **m** from the ciphertext c the following formula is used to get plain text m.  
   **m = cd mod n.**[1][2]

***Example :***

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 i: Select two large prime numbers, p, and q.

p = 7

q = 11

Step ii: 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 iii: 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.

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 the relative prime e of 60 as 7.

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

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

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

C = me mod n

C = 97 mod 77

C = 37

Step v: 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

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

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

m = cd mod n

m = 3743 mod 77

m = 9

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

1. **ROBBIN MILLER PRIMALITY TEST:**

The Miller-Rabin primality test is a probabilistic algorithm used to determine if a given number is a probable prime or definitely composite. It works based on the properties of Fermat's Little Theorem. The algorithm performs a series of modular exponentiation and probabilistic checks to assess whether a number is likely to be prime. It is widely used in practice due to its efficiency and effectiveness, even though it might have a small probability of error.

Here are the steps:

1. Input: Choose a candidate prime n to be tested for primality. Also, choose a parameter k that determines the accuracy of the test. A higher k value increases the accuracy but also the computational cost.
2. Factorization of n-1: Write n-1 as 2s.d, where s is the largest power of 2
3. dividing n-1, and d is an odd number.

iv) Witness Loop: Repeat the following steps k times

a. Choose a Witness: Select a random integer a such that 2 <= a <= n.

b. Compute ad mod n: Calculate x = ad mod n.

c. Check Conditions: If x = 1 mod n or x = -1 mod n, then continue to the next Iteration. Otherwise, proceed to the next step.

d. Square-and-Multiply: For r = 1 to s-1, compute x =x^2 mod n. If x = 1 mod n, then n is composite. If x = -1 mod n, break out of the loop.-

e. Final Check: If x != -1 mod n, then n is definitely composite.

v) Conclusion: If, after k iterations, n has passed all the tests, it is considered a probable prime. [2]

## **E. Advantages of RSA:**

* No Key Sharing: RSA encryption depends on using the receiver’s public key, so you don’t have to share any secret key to receive messages from others.
* Proof of Authenticity: Since the key pairs are related to each other, a receiver can’t intercept the message since they won’t have the correct private key to decrypt the information.
* Faster Encryption: The encryption process is faster than that of the DSA algorithm.
* Data Can’t Be Modified: Data will be tamper-proof in transit, since meddling with the data will alter the usage of the keys. And the private key won’t be able to decrypt the information, hence alerting the receiver of manipulation. [1]

**F.Analysis of RSA:**

**i). Brute-force analysis of RSA algorithm:**

The below table[Fig- ii.c] represent the complexity of RSA algorithm in different bit sizes.

|  |  |
| --- | --- |
| **Bit Size** | **Time taken (in milliseconds)** |
| **0** | **0** |
| **10** | **0.5** |
| **15** | **0.95** |
| **20** | **23.73** |
| **25** | **394.78** |
| **30** | **15985.57** |
| **31** | **29522.25** |
| **32** | **44630.68** |
| **33** | **110705.20** |
| **35** | **1184805.80** |

Fig- ii.c

**ii). Graphical Representation:**

The below graph[Fig- ii.d] represent the brute force attack strength of RSA with increasing bit size which is initially liinear the exponential. [2]

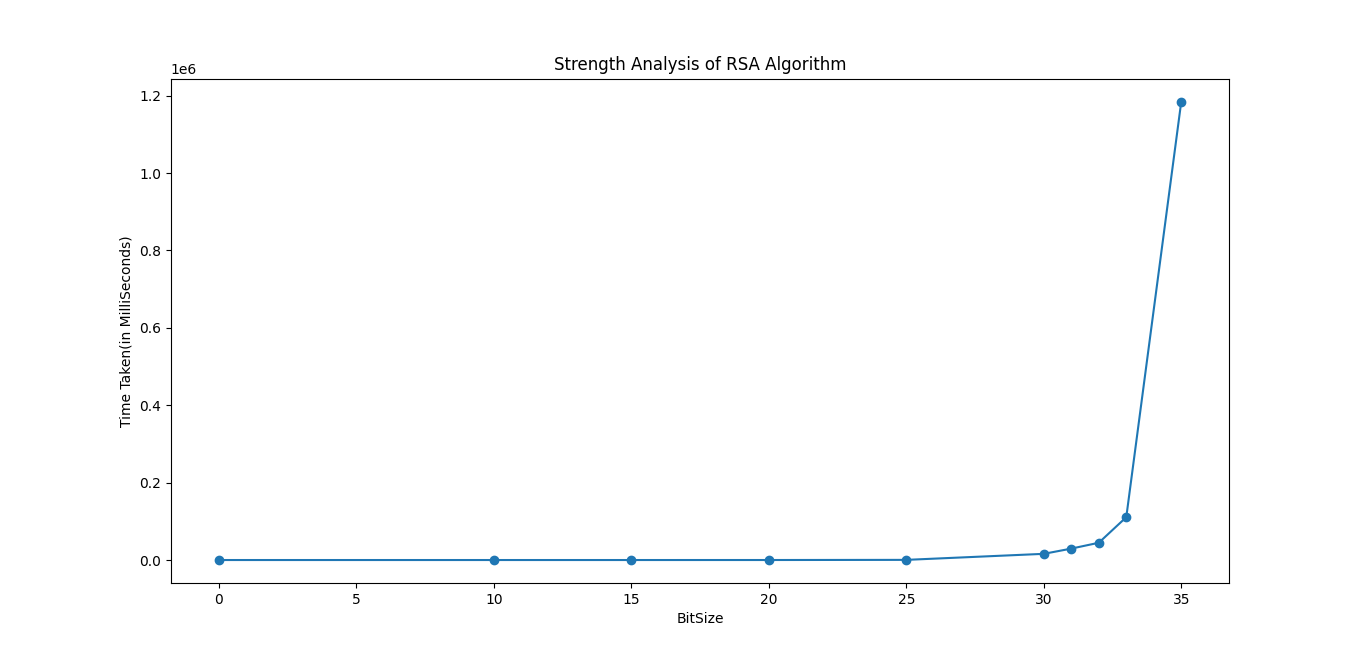
****

Fig- ii.d

1. **Conclusion**:

RSA (Rivest-Shamir-Adleman) is a foundational public-key cryptography algorithm that ensures secure data transmission. It involves generating a pair of keys: a public key for encryption and a private key for decryption, derived from the product of two large prime numbers. RSA's security relies on the computational difficulty of factoring large composite numbers. Despite its robustness, the algorithm requires careful implementation to prevent vulnerabilities, and the use of sufficiently large key sizes is crucial to maintain its security in the face of advancing computational capabilities.

1. **References:**

[1] The RSA Algorithm Evgeny Milanov 3 June 2009  
[2] Analysis of RSA based on Quantitating Key Security Strength

(Wenxue Tan Xiping Wang Xiaoping Lou Meisen Pan)

1. **Annexure:**
2. ***RSA CODE FOR ENCRYPTION AND DECRYPTION AND BRUITEFORRCE ANALYSIS IN JAVA:***

import java.util.\*;

import java.math.BigInteger;

import java.security.SecureRandom;

public class Main {

static Scanner sc = new Scanner(System.in);

BigInteger key1;

BigInteger key2;

BigInteger n;

int bitSize;

SecureRandom random = new SecureRandom();

public BigInteger generatePrime() {

bitSize = 35;// ?34

while (true) {

BigInteger randomBigInteger = new BigInteger(bitSize, random);

if (randomBigInteger.isProbablePrime(15)) {

return randomBigInteger;

}

}

}

public void setKeys() {

BigInteger prime1 = generatePrime();

BigInteger prime2 = generatePrime();

while (prime1.equals(prime2)) {

prime2 = generatePrime();

}

System.out.println("Prime 1 : " + prime1.toString());

System.out.println("Prime 2 : " + prime2.toString());

n = prime1.multiply(prime2);

// System.out.println("N: " + n.toString());

BigInteger phi = (prime1.subtract(new BigInteger("1"))).multiply(prime2.subtract(new BigInteger("1")));

BigInteger e = new BigInteger("2");

while (true) {

if ((e.gcd(phi)).compareTo(new BigInteger("1")) == 0)

break;

e = e.add(new BigInteger("1"));

}

// System.out.println("e : " + e.toString());

key1 = e;

BigInteger d = e.modInverse(phi);

// System.out.println("d : " + d.toString());

key2 = d;

}

public BigInteger encrypt(BigInteger message) {

return message.modPow(key1, n);

}

public BigInteger decrypt(BigInteger encryptedText) {

return encryptedText.modPow(key2, n);

}

public List<BigInteger> encoder(String message) {

List<BigInteger> msg = new ArrayList<>();

for (char letter : message.toCharArray()) {

BigInteger charValue = BigInteger.valueOf((long) letter);

BigInteger encryptedChar = encrypt(charValue);

msg.add(encryptedChar);

}

return msg;

}

public String decoder(List<BigInteger> encoded) {

StringBuilder s = new StringBuilder();

for (BigInteger encryptedChar : encoded) {

BigInteger decryptedChar = decrypt(encryptedChar);

s.append((char) decryptedChar.longValue());

}

return s.toString();

}

// !!!BREAK

public void StrengthTest() {

long ans = 0;

long ans1 = 0;

int i = 0;

while (i != 10) {

System.out.println((i + 1) + " : ");

setKeys();

String s = "Checking The Strength of Algorithm";

long startTime1 = System.currentTimeMillis();

List<BigInteger> v = encoder(s);

long endTime1 = System.currentTimeMillis();

long startTime = System.currentTimeMillis();

breakEncryption(n, key1, v, s);

long endTime = System.currentTimeMillis();

i++;

ans += (endTime - startTime);

ans1 += (endTime1 - startTime1);

}

System.out.println(bitSize + " --> " + (ans));

System.out.println(bitSize + " --> " + (ans1));

}

BigInteger decrypt(BigInteger encrpyted\_text, BigInteger key2, BigInteger n) {

BigInteger d = key2;

BigInteger decrypted = encrpyted\_text.modPow(d, n);

return decrypted;

}

public String decoder(List<BigInteger> encoded, BigInteger d, BigInteger n) {

StringBuilder s = new StringBuilder();

for (BigInteger encryptedChar : encoded) {

BigInteger decryptedChar = decrypt(encryptedChar, d, n);

s.append((char) decryptedChar.longValue());

}

return s.toString();

}

void breakEncryption(BigInteger n, BigInteger e, List<BigInteger> cipher, String txt) {

// ArrayList<BigInteger> pr = generatePrimes(n);

// long p = pr.get(0);

// long q = pr.get(1);

// for(BigInteger i=new BigInteger("2");)

// System.out.println("N: " + n.toString());

BigInteger p = new BigInteger("2");

while (p.compareTo(n) <= 0) {

if (n.remainder(p).equals(BigInteger.ZERO)) {

break;

// System.out.println("Prime divisor found: " + divisor);

// number = number.divide(divisor);

} else {

p = p.add(BigInteger.ONE);

}

}

// System.out.println("P : " + p.toString());

BigInteger q = n.divide(p);

// System.out.println("Q : " + q.toString());

// q = n.divide(p);

// System.out.println(p + " " + q);

// if (p == n || q == n) {

// System.out.println("Error");

// return;

// }

BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

BigInteger k = new BigInteger("1");

BigInteger d;

String s;

BigInteger nm;

while (true) {

nm = BigInteger.ONE.add(k.multiply(phi));

d = nm.divide(e);

// System.out.println("d : " + d);

s = decoder(cipher, d, n);

if (s.equals(txt)) {

// System.out.println(d + " " + k);

// System.out.println("Text : " + s);

break;

}

k = k.add(BigInteger.ONE);

}

}

public static void main(String[] args) {

Main rsa = new Main();

rsa.StrengthTest();

// rsa.setKeys();

// /\*

// \* String s = "Hi There";

// \* List<BigInteger> v = rsa.encoder(s);

// \* System.out.println("Encrypted Message : ");

// \* v.forEach((elem) -> System.out.print(elem + " "));

// \* System.out.println("\nDecrypted Message : ");

// \* String sh = rsa.decoder(v);

// \* System.out.println(sh);

// \*/

// // !Custom Check

// System.out.println("Enter Text to be Encrypted : ");

// String s = sc.nextLine();

// List<BigInteger> v = rsa.encoder(s);

// System.out.println("\nEncrypted Message : ");

// v.forEach((elem) -> System.out.print(elem + " "));

// System.out.println("\n\nDecrypted Message : ");

// String sh = rsa.decoder(v);

// rsa.StrengthTest(v);

// System.out.println(sh);

}

}