Implementation of RSA Algorithm

Network Security Assignment - CS1702

Submitted By
N Kathiravan (CS22B1036)

To

Dr. Narendran Rajagopalan
Associate Professor
Department of Computer Science and Engineering
National Institute of Technology Puducherry
Karaikal – 609609

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY
KARAIKAL – 609 609
April 2025

Introduction to RSA:

RSA (Rivest–Shamir–Adleman) is among the earliest public-key cryptographic systems and remains widely used for secure communication. In this type of system, the encryption key is public, while the decryption key is private and confidential. RSA relies on the computational difficulty of factoring large composite numbers — a challenge known as the "factoring problem" — to maintain its security. The name "RSA" comes from its inventors: Ron Rivest, Adi Shamir, and Leonard Adleman, who introduced the algorithm in 1977. Interestingly, a similar method was developed earlier in 1973 by Clifford Cocks, a British mathematician at GCHQ, but it remained classified until 1997.

An RSA user generates and shares a public key derived from two large secret prime numbers and an auxiliary value. While anyone can encrypt messages using this public key, only someone with knowledge of the original prime numbers can decrypt them. The difficulty of breaking RSA encryption is known as the "RSA problem," though it's still uncertain whether this problem is exactly equivalent to the factoring problem. So far, no effective attacks have been made public against RSA when strong, properly-sized keys are used.

Due to its slower performance, RSA is not typically used for direct encryption of large amounts of data. Instead, it is often employed to securely exchange keys for symmetric encryption systems, which are much faster for bulk data processing.

RSA Algorithm:

- Pick two large primes p, q.
- Compute $n= pq$ and $\phi(n)=\mathrm{lcm}\ (p-1, q-1)$
- Choose a public key e such that $1<e<\phi(n)$ and $\gcd\ (e, \phi(n)) =1$
- Calculate d such that $de\equiv1(\mathrm{mod}\ \phi(n))$
- Let the message key be: m
- Encrypt: $c \equiv m^{**}e\ (\mathrm{mod}\ n)$

- Decrypt: m ≡ c**d (mod n)

RSA Code and Explanation:

Imports:

```
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;
```

Greatest Common Divisor:

```

public class RSA {
    public static BigInteger gcd(BigInteger a, BigInteger b) {
        while (!b.equals(BigInteger.ZERO)) {
            BigInteger temp = b;
            b = a.mod(b);
            a = temp;
        }
        return a;
    }
```

Modular Inverse:

```
    public static BigInteger modInverse(BigInteger e, BigInteger phi) {
        return e.modInverse(phi);
    }
```

Prime Check:

```
    public static boolean isPrime(BigInteger n) {
        return n.isProbablePrime(10);
    }
```

Key Pair Generation:

```
    public static BigInteger[] generateKeypair(BigInteger p, BigInteger q) {
        if (!isPrime(p) || !isPrime(q)) {
            throw new IllegalArgumentException("Both numbers must be prime.");
        }
        if (p.equals(q)) {
            throw new IllegalArgumentException("p and q cannot be the same.");
        }

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

        BigInteger e = BigInteger.valueOf(2);
        while (e.compareTo(phi) < 0) {
            if (gcd(e, phi).equals(BigInteger.ONE)) {
                break;
            }
            e = e.add(BigInteger.ONE);
        }
        BigInteger d = modInverse(e, phi);
        return new BigInteger[] {e, n, d};
    }
```

Encryption:

```
    public static BigInteger[] encrypt(String message, BigInteger e,
BigInteger n) {
        BigInteger[] cipher = new BigInteger[message.length()];
        for (int i = 0; i < message.length(); i++) {
            cipher[i] = BigInteger.valueOf((int) message.charAt(i)).modPow(e,
n);
        }
        return cipher;
    }
```

Decrypt:

```
    public static String decrypt(BigInteger[] ciphertext, BigInteger d,
BigInteger n) {
        StringBuilder plaintext = new StringBuilder();
        for (BigInteger cipher : ciphertext) {
            plaintext.append((char) cipher.modPow(d, n).intValue());
        }
        return plaintext.toString();
    }
```

Main:

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first prime number (p): ");
        String input1 = scanner.nextLine();
        System.out.print("Enter second prime number (q): ");
        String input2 = scanner.nextLine();
        BigInteger p = new BigInteger(input1);
        BigInteger q = new BigInteger(input2);
        BigInteger[] keys = generateKeypair(p, q);
        BigInteger e = keys[0];
        BigInteger n = keys[1];
        BigInteger d = keys[2];
        System.out.print("Enter a message to encrypt: ");
        String message = scanner.nextLine();
        BigInteger[] ciphertext = encrypt(message, e, n);
        String decryptedMessage = decrypt(ciphertext, d, n);
        scanner.close();
        System.out.println("Original Message: " + message);
        System.out.println("Public Key: (e: " + e + ", n: " + n + ")");
        System.out.println("Private Key: (d: " + d + ", n: " + n + ")");
        System.out.println("Encrypted Message: ");
        for (BigInteger c : ciphertext) {
            System.out.print(c + " ");
        }
        System.out.println();
        System.out.println("Decrypted Message: " + decryptedMessage);
    }}
```

GUI:

Import:

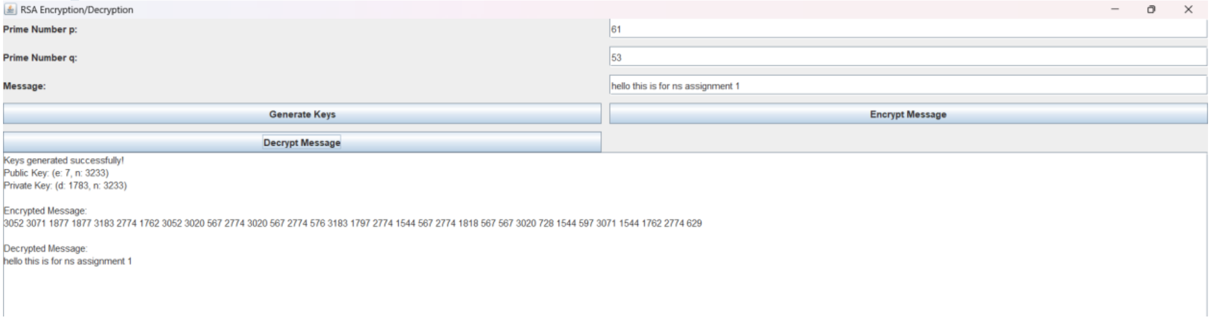
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.math.BigInteger;
```

GUI:

```
public class RSA_GUI extends JFrame {

    private JTextField pField, qField, messageField;
    private JTextArea outputArea;
    private JButton generateKeysButton, encryptButton, decryptButton;

    private BigInteger e, d, n;
    private BigInteger[] ciphertext;

    public RSA_GUI() {
        setTitle("RSA Encryption/Decryption");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 500);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridLayout(5, 2, 10, 10));
        pField = new JTextField();
        qField = new JTextField();
        messageField = new JTextField();
        generateKeysButton = new JButton("Generate Keys");
        encryptButton = new JButton("Encrypt Message");
        decryptButton = new JButton("Decrypt Message");

        inputPanel.add(new JLabel("Prime Number p:"));
        inputPanel.add(pField);
        inputPanel.add(new JLabel("Prime Number q:"));
        inputPanel.add(qField);
        inputPanel.add(new JLabel("Message:"));
        inputPanel.add(messageField);
        inputPanel.add(generateKeysButton);
        inputPanel.add(encryptButton);
        inputPanel.add(decryptButton);

        outputArea = new JTextArea();
        outputArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(outputArea);

        add(inputPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);

        // Button Listeners
        generateKeysButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                generateKeys();
            }
        });

        encryptButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                encryptMessage();
            }
        });

        decryptButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                decryptMessage();
            }
        });

        setVisible(true);
    }

    private void generateKeys() {
        try {
            BigInteger p = new BigInteger(pField.getText());
            BigInteger q = new BigInteger(qField.getText());

            BigInteger[] keys = RSA.generateKeypair(p, q);
            e = keys[0];
            n = keys[1];
            d = keys[2];

            outputArea.setText("");
            outputArea.append("Keys generated successfully!\n");
            outputArea.append("Public Key: (e: " + e + ", n: " + n + ")\n");
            outputArea.append("Private Key: (d: " + d + ", n: " + n + ")\n");

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Key Generation Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void encryptMessage() {
        try {
            if (e == null || n == null) {
                JOptionPane.showMessageDialog(this, "Please generate keys first!", "Error", JOptionPane.ERROR_MESSAGE);
```

```java
                return;
            }

            String message = messageField.getText();
            ciphertext = RSA.encrypt(message, e, n);

            outputArea.append("\nEncrypted Message:\n");
            for (BigInteger c : ciphertext) {
                outputArea.append(c.toString() + " ");
            }
            outputArea.append("\n");

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Encryption Error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void decryptMessage() {
        try {
            if (ciphertext == null || d == null || n == null) {
                JOptionPane.showMessageDialog(this, "Please encrypt a message
first!", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            String decrypted = RSA.decrypt(ciphertext, d, n);
            outputArea.append("\nDecrypted Message:\n" + decrypted + "\n");

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Decryption Error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new RSA_GUI());
    }
}
```

Output:



Prime Number p: 61

Prime Number q: 53

Message: hello this is for ns assignment 1

```
Keys generated successfully!
Public Key: (e: 7, n: 3233)
Private Key: (d: 1783, n: 3233)

Encrypted Message:
3052 3071 1877 1877 3183 2774 1762 3052 3020 567 2774 3020 567 2774 576 3183 1797 2774 1544 567 2774 1818 567 567 3020 728 1544 597 3071 1544 1762 2774 629

Decrypted Message:
hello this is for ns assignment 1
```