|  |  |
| --- | --- |
| **Academic Year: 2024-25** | **Programme: BTECH-Cyber (CSE)** |
| **Year: 2nd** | **Semester: IV** |
| **Student Name: Jal Bafana** | **Batch : K1** |
| **Roll No: K005** | **Date of experiment: 06.03.2025** |
| **Faculty: Rejo Mathew** | **Signature with Date:** |

**Experiment 9: Rabin Cryptosystem**

**Aim:** Write a program to implement Rabin cryptosystem.

**Learning Outcomes:**

After completion of this experiment, student should be able to

1. Describe working of Rabin cryptosystem.
2. Understand application of Rabin along with its advantage and limitations.

**Theory:**

Rabin Cryptosystem is a public-key cryptosystem invented by Michael Rabin. It uses asymmetric key encryption for communicating between two parties and encrypting the message. The security of Rabin cryptosystem is related to the difficulty of factorization. It has the advantage over the others that the problem on which it banks has proved to be hard as integer factorization. It has the disadvantage also, that each output of the Rabin function can be generated by any of four possible inputs. if each output is a ciphertext, extra complexity is required on decryption to identify which of the four possible inputs was the true plaintext.

**Procedure:**

**Key Generation**

1. Generate two very large prime numbers, p and q, which satisfies the condition  
   p ≠ q → p ≡ q ≡ 3 (mod 4)  
   For example:  
     p=139 and q=191
2. Calculate the value of n  
     n = p.q
3. Publish n as public key and save p and q as private key

**Encryption**

1. Get the public key n.
2. Convert the message to ASCII value. Then convert it to binary and extend the binary value with itself, and change the binary value back to decimal m.
3. Encrypt with the formula: C = m2 mod n
4. Send C to recipient.

**Decryption**

1. Accept C from sender.
2. Specify a and b with Extended Euclidean GCD such that, a.p + b.q = 1
3. Compute r and s using following formula:  
   r = C(p+1)/4 mod p  
   s = C(q+1)/4 mod q
4. Now, calculate X and Y using following formula:  
   X = ( a.p.s + b.q.r ) mod n  
   Y = ( a.p.s – b.q.r ) mod n
5. The four roots are,

m1=X, m2=-X, m3=Y, m4=-Y  
Now, Convert them to binary and divide them all in half.

1. Determine in which the left and right half are same. Keep that binary’s one half and convert it to decimal m. Get the ASCII character for the decimal value m. The resultant character gives the correct message sent by sender.

**Example 1**

1. Let p = 23 and q = 7
2. Bob selects p = 23 and q = 7. Note that both are congruent to 3 mod 4

i.e p and q are in the form 4k+3. p and q should not be equal

23 mod 4 = 3 and 7 mod 4 = 3

1. Bob calculate n= 23 \* 7 = 161, n (public key) = 161, where p and q are kept secret
2. Alice wants to send plaintext message = 45. Note that n and p are relatively prime

Alice calculates the C = m2 mod n = 45 2 mod 161 = 93

1. Alice sends C = 93 to Bob
2. Bob receives 93 and calculates 4 values

r = c (p+1)/4 mod p = (93 (23+1)/4) mod 23 = 1 mod 23 = 1

s = c (q+1)/4 mod q = (93 (7+1)/4) mod 7 = 4 mod 7 = 4

1. From the equation, a x 23 + b x 7 = 1, (-3 x 23 + 10 x 7 = 1) so a = -3 and b = 10
2. x = ((-3) x 23 x 4 + 10 x 7 x 1) mod 161 = (-276 + 70) mod 161 = 206 mod 161 = 45

y = ((-3) x 23 x 4 - 10 x 7 x 1) mod 161 = (-346) mod 161 = 24

**m1 = 45 = 101101**

m2 = 116 = 01110100

m3 = 24 = 011000

m4 = 137 = 10001001

**Example 2**

Given p = 11, q = 7, m = 45

Step 1: Public key = 77, For ciphertext = 452 mod 77 =23

Step 2: By Euclidean Algorithm a x p + b x q = 1, a = 2 and b = -3.

Step 3:

r = c (p+1)/4 mod p = 23 (11+1)/4 mod 11 = 1

s = c (q+1)/4 mod q = 23 (7+1)/4 mod 11 = 4

x = (2 x 11 x 4 + (-3) x 7 x 1) mod 77 = 67

y = (2 x 11 x 4 - (-3) x 7 x 1) mod 77 = 32

Step 4: Roots

m1 = X = 67 = 1000011

m2 = - X = (77-67) =10 = 1000011

m3 = Y = 32 = 100000

**m4 = -Y = (77-32) = 45 = 101101**

**Code: *type or copy your completed working code here***

***Code 1:***

*import sympy*

*# Set up parameters*

*prime1 = 11*

*prime2 = 7*

*modulus = prime1 \* prime2*

*print("Public key modulus:", modulus)*

*# Input plaintext number to encrypt*

*plaintext = int(input("Enter number to encrypt: "))*

*print("Plaintext message:", plaintext)*

*# Encryption: compute ciphertext = plaintext^2 mod modulus*

*ciphertext = (plaintext \* plaintext) % modulus*

*print("Encrypted ciphertext:", ciphertext)*

*# Decryption: compute r and s*

*r\_value = pow(ciphertext, (prime1 + 1) // 4, prime1)*

*s\_value = pow(ciphertext, (prime2 + 1) // 4, prime2)*

*print("r\_value:", r\_value)*

*print("s\_value:", s\_value)*

*# Use known coefficients from Extended Euclidean Algorithm:*

*def find\_coefficients(prime1, prime2):*

*coeff1, coeff2, \_ = sympy.gcdex(prime1, prime2)*

*return coeff1, coeff2*

*coeff1, coeff2 = find\_coefficients(prime1, prime2)*

*print("coeff1:", coeff1, "coeff2:", coeff2)*

*# Compute X and Y*

*X\_value = (coeff1 \* prime1 \* s\_value + coeff2 \* prime2 \* r\_value) % modulus*

*Y\_value = (coeff1 \* prime1 \* s\_value - coeff2 \* prime2 \* r\_value) % modulus*

*print("X\_value:", X\_value, "Y\_value:", Y\_value)*

*# The four possible plaintext values are:*

*possible\_plaintext1 = X\_value*

*possible\_plaintext2 = (-X\_value) % modulus*

*possible\_plaintext3 = Y\_value*

*possible\_plaintext4 = (-Y\_value) % modulus*

*print("Possible plaintext values:", possible\_plaintext1, possible\_plaintext2, possible\_plaintext3, possible\_plaintext4)*

*for value in [possible\_plaintext1, possible\_plaintext2, possible\_plaintext3, possible\_plaintext4]:*

*binary\_str = bin(value)[2:]*

*if binary\_str == binary\_str[::-1]:*

*print("The result is:", value)*

**Code 2:**

*import sympy*

*# Key Generation*

*p = 23*

*q = 7*

*n = p \* q*

*print("Public Key (n):", n)*

*print("Private Key (p, q):", (p, q))*

*# Encryption*

*message = 45*

*print("Original Message:", message)*

*ciphertext = (message \* message) % n*

*print("Encrypted Ciphertext:", ciphertext)*

*# Decryption*

*# Calculating r and s*

*r = pow(ciphertext, (p + 1) // 4, p)*

*s = pow(ciphertext, (q + 1) // 4, q)*

*print("R = ", r)*

*print("S = ", s)*

*# Extended Euclidean Algorithm to find a and b such that a \* p + b \* q = 1*

*def find\_a\_b(p, q):*

*\_, a, b = sympy.gcdex(p, q)*

*return a, b*

*# Using sympy.gcdex to find a and b*

*a, b = find\_a\_b(p, q)*

*print("a =", a)*

*print("b =", b)*

*# Calculate X and Y*

*X = (a \* p \* s + b \* q \* r) % n*

*Y = (a \* p \* s - b \* q \* r) % n*

*# The four roots are m1, m2, m3, m4*

*m1 = X*

*m2 = -X % n*

*m3 = Y*

*m4 = -Y % n*

*print("Possible decrypted messages (m1, m2, m3, m4):", (m1, m2, m3, m4))*

*# Check the binary representations of the four possible messages*

*def binary\_to\_decimal(binary\_str):*

*return int(binary\_str, 2)*

*def check\_binary\_root(root):*

*# Convert the integer to binary string*

*binary\_str = bin(root)[2:]*

*# Check if the left and right halves are identical*

*mid = len(binary\_str) // 2*

*left\_half = binary\_str[:mid]*

*right\_half = binary\_str[mid:]*

*if left\_half == right\_half:*

*return binary\_to\_decimal(left\_half)*

*return None*

*# Try to find the correct message from the decrypted roots*

*for root in (m1, m2, m3, m4):*

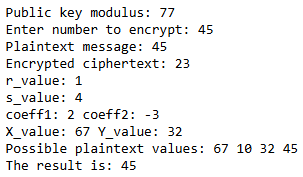
*possible\_message = check\_binary\_root(root)*

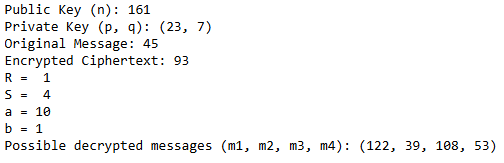
*if possible\_message is not None:*

*print(f"Correct decrypted message is: {possible\_message}")*

*break*

**Output:**

****

****

**Questions:**

1. What is the one-way trapdoor function used in the Rabin cryptosystem?

The one-way trapdoor function in Rabin is squaring the message mmm modulo n=p×qn = p \times qn=p×q, where ppp and qqq are primes. It’s easy to compute but hard to reverse without knowing the factors of nnn.

1. How does the Rabin cryptosystem differ from **RSA** in terms of security and computational efficiency?

Rabin is more secure than RSA due to fewer possible plaintexts but is computationally faster since it uses squaring, not modular exponentiation. However, Rabin has four possible plaintexts for each ciphertext.

1. Why is the Rabin cryptosystem classified as a **probabilistic asymmetric encryption algorithm?**

It’s probabilistic because each ciphertext can correspond to multiple plaintexts, making decryption involve checking multiple possibilities.

1. What are some potential **attacks** against the Rabin cryptosystem, and how can they be mitigated?

Potential attacks include chosen ciphertext attacks and message guessing. These can be mitigated by adding padding or using additional checks to validate plaintexts.

**Conclusion:** *The Rabin cryptosystem is efficient and secure but introduces complexity with multiple possible plaintexts per ciphertext. This experiment highlights its practical and theoretical aspects compared to other systems like RSA.*