| **Course Name:** | **Applied Cryptography (**116U01E628**)** | **Semester:** | **VI** |
| --- | --- | --- | --- |
| **Date of Performance:** | **26/01/2025** | **DIV/ Batch No:** | **C - 3** |
| **Student Name:** | **Romil Lodaya** | **Roll No:** | **16010122096** |

**Experiment No: 4**

**Title: Cryptographic Arithmetic**

| **Aim and Objective of the Experiment:** |
| --- |
| To learn about Cryptographic Arithmetic and its implementation   1. Prime number generation 2. Primality test |

| **COs to be achieved:** |
| --- |
| **CO2: Demonstrate and implement various Cryptographic Algorithms for securing systems** |

| **Books/ Journals/ Websites referred:** |
| --- |
| 1. Stallings, W., Cryptography and Network Security: Principles and Practice, Second edition, Person Education 2. “Caesar Cipher in cryptography”, <https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/>, last retrieved on Aug 01, 2023 3. “PlayFair Cipher in cryptography”: <https://www.geeksforgeeks.org/playfair-cipher-with-examples/>, last retrieved on Aug 01, 2023 4. “Transposition cipher in cryptology,  ”<https://www.britannica.com/topic/transposition-cipher>, last retrieved on Aug 01, 2023 |

| **Theory:** |
| --- |
| **Abstract**:-  Cryptographic arithmetic involves using mathematical operations and techniques within cryptography to secure communication and data. It includes concepts like modular arithmetic, one-way functions, prime numbers, exponentiation, and elliptic curve cryptography, all of which are crucial for ensuring the confidentiality, integrity, and authenticity of data in cryptographic systems.  **Related Theory:**  **1. Number Theory:** Number theory forms the foundation of many cryptographic algorithms. Concepts such as prime numbers, modular arithmetic, greatest common divisors, and Euler's totient function are central to understanding and implementing cryptographic arithmetic.  **2. Modular Arithmetic:** Modular arithmetic is fundamental in cryptographic operations. It involves performing arithmetic operations within a finite set, called a modulus. Theorems related to modular arithmetic, like Euler's Totient Theorem, are critical for cryptographic algorithms like RSA and Diffie-Hellman.  **3. Additive Inverse:** Two numbers a and b are additive inverses of each other if a + b ≡ 0 (mod n). The additive inverse of a can be calculated as b = n − a.  **4. Multiplicative Inverse:** Two numbers a and b are the multiplicative inverse of each other if a × b ≡ 1 (mod n) |

| **Algorithm :** |
| --- |
| **Algorithm:**   1. **Function checkPrime(n)**:    * If n == 1, return true (optional, depending on whether 1 is considered prime or not).    * Initialize cnt = 0.    * Iterate i from 1 to sqrt(n):      + If n % i == 0:        - Increment cnt.        - If n / i != i, increment cnt again.    * If cnt == 2, return true (prime); else, return false. 2. **Main**:    * Take input a and b (range limits).    * Print "Prime numbers between a and b:".    * Iterate i from a to b:      + If checkPrime(i) returns true, print i. |

| **Solve a small numerical for assigned algorithm(Paste photograph of same) :** |
| --- |
|  |

| **Code :** |
| --- |
| **1] Prime Number Generation & Primality Test:**  #include<bits/stdc++.h>  using namespace std;  bool checkPrime(int n)  {  if(n == 1)return true;    int cnt = 0;  for(int i = 1; i <= sqrt(n); i++)  {  if(n % i == 0)  {  cnt++;  if(n / i != i)  {  cnt++;  }  }  }  if(cnt == 2)return true;  else return false;  }  int main()  {  int a, b;  cout << "Enter the range (a & b): ";  cin >> a >> b;    cout << "Prime number between " << a << " & " << b << ": " << endl;  for(int i = a; i <= b; ++i)  {  if(checkPrime(i))  {  cout << i << " ";  }  }    return 0;  } |

| **Output:** |
| --- |
|  |

| **Post Lab Subjective/Objective type Questions:** |
| --- |
| 1. What is the purpose of primality testing in cryptography?   **1. Purpose of Primality Testing in Cryptography**  Primality testing is a fundamental process in cryptography, particularly in **public-key cryptography**. Here’s why it’s important:  **a. Key Generation:**   * Cryptographic systems like **RSA** rely on the use of **large prime numbers** to generate public and private keys. * During key generation:   + Two large prime numbers, p*p* and q*q*, are selected.   + Their product, n=p⋅q*n*=*p*⋅*q*, is used as part of the public key.   + The security of the system depends on the difficulty of factoring n*n* back into p*p* and q*q*.   **b. Ensuring Security:**   * The strength of cryptographic algorithms like RSA depends on the **hardness of factoring large composite numbers**. * If the primes used are not truly prime (e.g., composite), the system becomes vulnerable to attacks. * Primality testing ensures that the numbers chosen for key generation are indeed prime, maintaining the security of the system.   **c. Efficient Algorithms:**   * Primality tests like the **Miller-Rabin test** or **AKS primality test** are used to efficiently check whether a number is prime. * These tests are probabilistic or deterministic and are optimized for large numbers, making them suitable for cryptographic applications.   **d. Applications:**   * Primality testing is used in:   + Generating RSA keys.   + Creating secure communication protocols (e.g., SSL/TLS).   + Digital signatures and authentication mechanisms.  1. Explain how the Chinese Remainder Theorem can be used to solve a system of simultaneous congruences.   The **Chinese Remainder Theorem (CRT)** is a powerful mathematical tool used to solve systems of simultaneous congruences. Here’s how it works and its applications:  **a. Problem Statement:**  Given a system of congruences:  x≡a1 (mod m1)x≡a2 (mod m2)⋮x≡an (mod mn)*x*≡*a*1​ (mod *m*1​)*x*≡*a*2​ (mod *m*2​)⋮*x*≡*an*​ (mod *mn*​)  where m1,m2,…,mn*m*1​,*m*2​,…,*mn*​ are **pairwise coprime** (i.e., gcd⁡(mi,mj)=1gcd(*mi*​,*mj*​)=1 for all i≠j*i*=*j*), the CRT guarantees a **unique solution modulo M=m1⋅m2⋅⋯⋅mn*M*=*m*1​⋅*m*2​⋅⋯⋅*mn*​**.  **b. Steps to Solve Using CRT:**   1. **Compute M*M*:**    * Calculate M=m1⋅m2⋅⋯⋅mn*M*=*m*1​⋅*m*2​⋅⋯⋅*mn*​. 2. **Compute Mi*Mi*​ for each i*i*:**    * For each modulus mi*mi*​, compute Mi=M/mi*Mi*​=*M*/*mi*​. 3. **Find Modular Inverses yi*yi*​:**    * For each Mi*Mi*​, compute its modular inverse yi*yi*​ such that:   Mi⋅yi≡1 (mod mi)*Mi*​⋅*yi*​≡1 (mod *mi*​)   * + This inverse exists because Mi*Mi*​ and mi*mi*​ are coprime.  1. **Compute the Solution x*x*:**    * The solution is given by:   x=∑i=1nai⋅Mi⋅yi (mod M)*x*=*i*=1∑*n*​*ai*​⋅*Mi*​⋅*yi*​ (mod *M*)  **c. Example:**  Solve the system:  x≡2 (mod 3)x≡3 (mod 5)x≡2 (mod 7)*x*≡2 (mod 3)*x*≡3 (mod 5)*x*≡2 (mod 7)   * Step 1: M=3⋅5⋅7=105*M*=3⋅5⋅7=105. * Step 2: M1=105/3=35*M*1​=105/3=35, M2=105/5=21*M*2​=105/5=21, M3=105/7=15*M*3​=105/7=15. * Step 3: Find inverses:   + 35⋅y1≡1 (mod 3)⇒y1=235⋅*y*1​≡1 (mod 3)⇒*y*1​=2.   + 21⋅y2≡1 (mod 5)⇒y2=121⋅*y*2​≡1 (mod 5)⇒*y*2​=1.   + 15⋅y3≡1 (mod 7)⇒y3=115⋅*y*3​≡1 (mod 7)⇒*y*3​=1. * Step 4: Compute x*x*:   x=(2⋅35⋅2)+(3⋅21⋅1)+(2⋅15⋅1) (mod 105)x=140+63+30 (mod 105)x=233 (mod 105)x=23*x*=(2⋅35⋅2)+(3⋅21⋅1)+(2⋅15⋅1) (mod 105)*x*=140+63+30 (mod 105)*x*=233 (mod 105)*x*=23   * The solution is x=23*x*=23.   **d. Applications in Cryptography:**   * **RSA Decryption:**   + CRT is used to speed up RSA decryption by breaking the computation into smaller, parallelizable steps using the prime factors p*p* and q*q*. * **Error Correction:**   + CRT is used in coding theory to correct errors in data transmission. * **Secret Sharing:**   + CRT is used in schemes like **Shamir’s Secret Sharing** to distribute and reconstruct secrets. |

| **Conclusion:** |
| --- |
| This experiment enhanced understanding of cryptographic arithmetic by implementing prime number generation and primality testing, achieving CO2 through foundational cryptographic techniques. |