SEMESTER PROJECT

CRYPTOGRAPHY AND NETWORK SECURITY LAB

Design & Implementation a Asymmetric Cipher based Digital Signature+ Confidentiality Crypto System

Implement RSA based Digital Signature+Confidentiality cryptosystem as follows:-

Using mod n=55, Generate asymmetric cipher based key pairs for Sender

(Alice: Apub, Apvt)) & recipient (Bob: Bpub, Bpvt)

1) **Using mod n=55, Generate asymmetric cipher based key pairs for Sender (Alice: Apub, Apvt)) & recipient (Bob: Bpub, Bpvt).**

2) **Give block diagram to implement Asymmetric (RSA) based Confidentiality crypto-System (sender Alice, rcvr Bob), use key names of step1. Hence for plaintext= 'd' , compute Ciphertext at Sender. Feed this ciphertext at rcvr side to recover plaintext.**

3) **Give block diagram to implement Asymmetric (RSA) based Digital Signature crypto-System (sender Alice, rcvr Bob), use key names of step1. Hence for plaintext= 'd' , compute Signature algorithm output at Sender. Feed this output to rcvr side to verify/ authenticate sender at rcvr.**

4) **Give block diagram to implement Asymmetric (RSA) based combined Digita Signature + Confidentiality crypto-System (sender Alice, rcvr Bob), use key names of step1. Hence for plaintext= 'd' , compute final output at Sender. Feed this output to rcvr side to verify/ recover plaintext at rcvr.**

C++ Code For Key Generation :-

Public Key (e , d) :

Private Key (d) :

```cpp
#include <iostream>
#include <cmath>

// Function to check if two numbers are coprime (i.e., their GCD is 1)
bool isCoprime(int a, int b) {
	while (b != 0) {
		int temp = b;
		b = a % b;
		a = temp;
	}
	return a == 1;
}

// Function to calculate the modular inverse using Extended Euclidean Algorithm
int modInverse(int a, int m) {
	int m0 = m, t, q;
	int x0 = 0, x1 = 1;

	if (m == 1)
		return 0;

	while (a > 1) {
		// q is quotient
		q = a / m;
		t = m;

		// m is remainder now; process same as Euclid's algo
		m = a % m, a = t;

		t = x0;

		x0 = x1 - q * x0;

		x1 = t;
	}

	// Make x1 positive
	if (x1 < 0)
		x1 += m0;

	return x1;
}

int main() {
	// Given modulus 'n' for RSA cipher
	int n = 55;

	// Calculate pi(n) for n=55 (pi(n) is Euler's totient function for n)
	int pi_n = 0;
	for (int i = 1; i < n; i++) {
		if (isCoprime(i, n)) {
```

```cpp
                pi_n++;
            }
        }

        // Find public key 'e' such that it is the smallest coprime to pi(n)'
        int e = 0;
        for (int i = 2; i < pi_n; i++) {
            if (isCoprime(i, pi_n)) {
                e = i;
                break;
            }
        }

        // Assuming sender and receiver have the same public and private keys
        int d = modInverse(e, pi_n);

        // Display the results
        std::cout << "Public key (e, n): (" << e << ", " << n << ")" << std::endl;
        std::cout << "Private key (d): " << d << std::endl;

        return 0;
}
```

Microsoft Visual Studio Debug Console

```
Public key (e, n): (3, 55)
Private key (d): 27

C:\Users\toshiba\source\repos\Project19\De
To automatically close the console when de
Press any key to close this window . . .
```

C++ Code for Asymmetric RSA based Confidentially System :

```cpp
#include <iostream>
#include <cmath>

// Function prototype for powerMod
int powerMod(int a, int b, int m);

// Function to check if two numbers are coprime (i.e., their GCD is 1)
bool isCoprime(int a, int b) {
        while (b != 0) {
                int temp = b;
                b = a % b;
                a = temp;
        }
        return a == 1;
}

// Function to calculate the modular inverse using Extended Euclidean Algorithm
int modInverse(int a, int m) {
```

```
	int m0 = m, t, q;
	int x0 = 0, x1 = 1;

	if (m == 1)
		return 0;

	while (a > 1) {
		// q is quotient
		q = a / m;
		t = m;

		// m is remainder now; process same as Euclid's algo
		m = a % m, a = t;

		t = x0;

		x0 = x1 - q * x0;

		x1 = t;
	}

	// Make x1 positive
	if (x1 < 0)
		x1 += m0;

	return x1;
}

int main() {
	// Given modulus 'n' for RSA cipher
	int n;
	std::cout << "Enter the Modulus Value : ";
	std::cin >> n;

	// Calculate pi(n) for n=55 (pi(n) is Euler's totient function for n)
	int pi_n = 0;
	for (int i = 1; i < n; i++) {
		if (isCoprime(i, n)) {
			pi_n++;
		}
	}

	// Find public key 'e' such that it is the smallest coprime to pi(n)'
	int e = 0;
	for (int i = 2; i < pi_n; i++) {
		if (isCoprime(i, pi_n)) {
			e = i;
			break;
		}
	}

	// Assuming sender and receiver have the same public and private keys
	int d = modInverse(e, pi_n);

	// Display the results
	std::cout << "Public key (e, n): (" << e << ", " << n << ")" << std::endl;
	std::cout << "Private key (d): " << d << std::endl;
```

```cpp
    // Input the plaintext letter (single letter)
    char alphabet[26] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm',
                              'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z' };
    char plaintext;
    std::cout << "Enter a single letter as plaintext: ";
    std::cin >> plaintext;

    // Find the value of the plaintext letter using the alphabet array
    int plaintextValue = -1;
    for (int i = 0; i < 26; i++) {
        if (alphabet[i] == plaintext) {
            plaintextValue = i;
            break;
        }
    }

    if (plaintextValue == -1) {
        std::cout << "Invalid input. Please enter a valid letter." << std::endl;
        return 1;
    }

    // Encryption: C = P^e (mod n)
    int ciphertext = powerMod(plaintextValue, e, n);

    // Decryption: P = C^d (mod n)
    int decryptedValue = powerMod(ciphertext, d, n);

    // Check if the decryption is valid
    if (decryptedValue >= 0 && decryptedValue < 26) {
        char decryptedLetter = alphabet[decryptedValue];
        std::cout << "Encrypted ciphertext (C): " << ciphertext << std::endl;
        std::cout << "Decrypted plaintext (P): " << decryptedLetter << std::endl;
    }
    else {
        std::cout << "Decryption failed. Please check the keys and input." <<
std::endl;
    }

    return 0;
}

// Definition of the powerMod function
int powerMod(int a, int b, int m) {
    int result = 1;
    a = a % m;

    while (b > 0) {
        if (b & 1)
            result = (result * a) % m;

        b = b >> 1;
        a = (a * a) % m;
    }

    return result;
}
```

Microsoft Visual Studio Debug Console

```
Enter the Modulus Value : 55
Public key (e, n): (3, 55)
Private key (d): 27
Enter a single letter as plaintext: d
Encrypted ciphertext (C): 27
Decrypted plaintext (P): d
```

C ++ Code For RSA Based Digital Signature System :-

```
#include <iostream>
#include <cmath>

// Function prototype for powerMod
int powerMod(int a, int b, int m);

// Function to check if two numbers are coprime (i.e., their GCD is 1)
bool isCoprime(int a, int b) {
	while (b != 0) {
		int temp = b;
		b = a % b;
		a = temp;
	}
	return a == 1;
}

// Function to calculate the modular inverse using Extended Euclidean Algorithm
int modInverse(int a, int m) {
	int m0 = m, t, q;
	int x0 = 0, x1 = 1;

	if (m == 1)
		return 0;

	while (a > 1) {
		// q is quotient
		q = a / m;
		t = m;

		// m is remainder now; process same as Euclid's algo
		m = a % m, a = t;

		t = x0;

		x0 = x1 - q * x0;

		x1 = t;
	}

	// Make x1 positive
```

```cpp
        if (x1 < 0)
            x1 += m0;

        return x1;
    }

    int main() {
        // Given modulus 'n' for RSA cipher
        int n;
        std::cout << "Enter the Modulus Value : ";
        std::cin >> n;

        // Calculate pi(n) for n=55 (pi(n) is Euler's totient function for n)
        int pi_n = 0;
        for (int i = 1; i < n; i++) {
            if (isCoprime(i, n)) {
                pi_n++;
            }
        }

        // Find public key 'e' such that it is the smallest coprime to pi(n)'
        int e = 0;
        for (int i = 2; i < pi_n; i++) {
            if (isCoprime(i, pi_n)) {
                e = i;
                break;
            }
        }

        // Assuming sender and receiver have the same public and private keys
        int d = modInverse(e, pi_n);

        // Display the results
        std::cout << "Public key (e, n): (" << e << ", " << n << ")" << std::endl;
        std::cout << "Private key (d): " << d << std::endl;

        // Input the plaintext letter (single letter)
        char alphabet[26] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
    'm',
                                              'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
    'w', 'x', 'y', 'z' };
        char plaintext;
        std::cout << "Enter a single letter as plaintext: ";
        std::cin >> plaintext;

        // Find the value of the plaintext letter using the alphabet array
        int plaintextValue = -1;
        for (int i = 0; i < 26; i++) {
            if (alphabet[i] == plaintext) {
                plaintextValue = i;
                break;
            }
        }

        if (plaintextValue == -1) {
            std::cout << "Invalid input. Please enter a valid letter." << std::endl;
            return 1;
        }
```

```
        // Encryption: C = P^d (mod n)
        int ciphertext = powerMod(plaintextValue, d, n);

        // Decryption: P = C^e (mod n)
        int decryptedValue = powerMod(ciphertext, e, n);

        // Check if the decryption is valid

        char decryptedLetter = alphabet[decryptedValue];
        std::cout << "Encrypted ciphertext (C): " << ciphertext << std::endl;
        std::cout << "Decrypted plaintext (P): " << decryptedLetter << std::endl;



        return 0;
}

// Definition of the powerMod function
int powerMod(int a, int b, int m) {
        int result = 1;
        a = a % m;

        while (b > 0) {
                if (b & 1)
                        result = (result * a) % m;

                b = b >> 1;
                a = (a * a) % m;
        }

        return result;
}
```

Microsoft Visual Studio Debug Console

```
Enter the Modulus Value : 55
Public key (e, n): (3, 55)
Private key (d): 27
Enter a single letter as plaintext: d
Encrypted ciphertext (C): 42
Decrypted plaintext (P): d
```

C ++ Code For RSA Based Combined Digital Signature + Confidentially System :-

```
#include <iostream>
#include <cmath>

// Function prototype for powerMod
int powerMod(int a, int b, int m);

// Function to check if two numbers are coprime (i.e., their GCD is 1)
bool isCoprime(int a, int b) {
        while (b != 0) {
```

```
			int temp = b;
			b = a % b;
			a = temp;
		}
		return a == 1;
	}

	// Function to calculate the modular inverse using Extended Euclidean Algorithm
	int modInverse(int a, int m) {
		int m0 = m, t, q;
		int x0 = 0, x1 = 1;

		if (m == 1)
			return 0;

		while (a > 1) {
			// q is quotient
			q = a / m;
			t = m;

			// m is remainder now; process same as Euclid's algo
			m = a % m, a = t;

			t = x0;

			x0 = x1 - q * x0;

			x1 = t;
		}

		// Make x1 positive
		if (x1 < 0)
			x1 += m0;

		return x1;
	}

	// Function to perform RSA encryption with confidentiality and digital signature
	std::pair<int, int> rsaEncrypt(int plaintext, int e, int d, int n) {
		// Encryption: T = P^d (mod n)
		int T = powerMod(plaintext, d, n);

		// Digital Signature: C = T^e (mod n)
		int C = powerMod(T, e, n);

		return std::make_pair(T, C);
	}

	// Function to perform RSA decryption with confidentiality and digital signature
	int rsaDecrypt(std::pair<int, int> encryptedData, int e, int d, int n) {
		int T = encryptedData.first;
		int C = encryptedData.second;

		// Digital Signature Verification: T = C^d (mod n)
		int verifiedT = powerMod(C, d, n);

		// Confidentiality Decryption: P = T^e (mod n)
		int decryptedPlaintext = powerMod(verifiedT, e, n);
```

```
        return decryptedPlaintext;
}

// Definition of the powerMod function
int powerMod(int a, int b, int m) {
        int result = 1;
        a = a % m;

        while (b > 0) {
                if (b & 1)
                        result = (result * a) % m;

                b = b >> 1;
                a = (a * a) % m;
        }

        return result;
}

int main() {
        // Given modulus 'n' for RSA cipher
        int n;
        std::cout << "Enter the Modulus Value : ";
        std::cin >> n;

        // Calculate pi(n) for n=55 (pi(n) is Euler's totient function for n)
        int pi_n = 0;
        for (int i = 1; i < n; i++) {
                if (isCoprime(i, n)) {
                        pi_n++;
                }
        }

        // Find public key 'e' such that it is the smallest coprime to pi(n)'
        int e = 0;
        for (int i = 2; i < pi_n; i++) {
                if (isCoprime(i, pi_n)) {
                        e = i;
                        break;
                }
        }

        // Assuming sender and receiver have the same public and private keys
        int d = modInverse(e, pi_n);

        // Display the results
        std::cout << "Public key (e, n): (" << e << ", " << n << ")" << std::endl;
        std::cout << "Private key (d): " << d << std::endl;

        // Input the plaintext letter (single letter)
        char alphabet[26] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm',
                                                'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z' };
        char plaintext;
        std::cout << "Enter a single letter as plaintext: ";
        std::cin >> plaintext;
```

```cpp
	// Find the value of the plaintext letter using the alphabet array
	int plaintextValue = -1;
	for (int i = 0; i < 26; i++) {
		if (alphabet[i] == plaintext) {
			plaintextValue = i;
			break;
		}
	}

	if (plaintextValue == -1) {
		std::cout << "Invalid input. Please enter a valid letter." << std::endl;
		return 1;
	}

	// Encryption with confidentiality and digital signature
	std::pair<int, int> encryptedData = rsaEncrypt(plaintextValue, e, d, n);

	// Decryption with confidentiality and digital signature
	int decryptedPlaintext = rsaDecrypt(encryptedData, e, d, n);

	// Check if the decryption is valid
	if (decryptedPlaintext >= 0 && decryptedPlaintext < 26) {
		char decryptedLetter = alphabet[decryptedPlaintext];
		std::cout << "Encrypted ciphertext (C): " << encryptedData.second <<
std::endl;
		std::cout << "Decrypted plaintext (P): " << decryptedLetter << std::endl;
	}
	else {
		std::cout << "Decryption failed. Please check the keys and input." <<
std::endl;
	}

	return 0;
}
```

Microsoft Visual Studio Debug Console

```
Enter the Modulus Value : 55
Public key (e, n): (3, 55)
Private key (d): 27
Enter a single letter as plaintext: d
Encrypted ciphertext (C): 3
Decrypted plaintext (P): d
```