**CRYPTOGRAPY LAB PROGRAMS**

M.YUGANDHAR REDDY  
REG.NO:192372016

1. **1. Write a program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in reverse order. Design a key-generation scheme with the appropriate shift schedule for the decryption process.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/des.h>

void des\_decrypt(const unsigned char \*ciphertext, int len, const unsigned char \*key, unsigned char \*decrypted) {

DES\_key\_schedule key\_schedule;

DES\_set\_key((DES\_cblock \*)key, &key\_schedule);

DES\_ecb\_encrypt((DES\_cblock \*)ciphertext, (DES\_cblock \*)decrypted, &key\_schedule, DES\_DECRYPT);

}

int main() {

unsigned char key[8] = "8bytekey";

unsigned char encrypted\_text[8] = { /\* Replace with actual encrypted data \*/ };

unsigned char decrypted\_text[8];

des\_decrypt(encrypted\_text, sizeof(encrypted\_text), key, decrypted\_text);

printf("Decrypted Text: %s\n", decrypted\_text);

return 0;

}

1. **Write a program for encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Both of which follow from the definition of CBC. Which of the two would you choose:**

**For security? b. For performance?**

**#**include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/aes.h>

#include <openssl/rand.h>

void aes\_encrypt(const unsigned char \*plaintext, int len, const unsigned char \*key, unsigned char \*ciphertext, unsigned char \*iv) {

AES\_KEY encrypt\_key;

AES\_set\_encrypt\_key(key, 256, &encrypt\_key);

AES\_cbc\_encrypt(plaintext, ciphertext, len, &encrypt\_key, iv, AES\_ENCRYPT);

}

int main() {

unsigned char key[32] = "thisisaverysecurekeyof32bytes!";

unsigned char iv[AES\_BLOCK\_SIZE];

RAND\_bytes(iv, sizeof(iv));

unsigned char plaintext[] = "Sensitive Data";

unsigned char ciphertext[sizeof(plaintext) + AES\_BLOCK\_SIZE];

aes\_encrypt(plaintext, sizeof(plaintext), key, ciphertext, iv);

printf("Encrypted Text: ");

for (int i = 0; i < sizeof(plaintext); i++) {

printf("%02x", ciphertext[i]);

}

printf("\n");

return 0;

}

1. **Write a  program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed?**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/aes.h>

#include <openssl/rand.h>

void pad\_message(unsigned char \*plaintext, int \*len) {

int pad\_len = AES\_BLOCK\_SIZE - (\*len % AES\_BLOCK\_SIZE);

for (int i = 0; i < pad\_len; i++) {

plaintext[\*len + i] = (i == 0) ? 0x80 : 0x00;

}

\*len += pad\_len;

}

void aes\_encrypt\_ecb(const unsigned char \*plaintext, int len, const unsigned char \*key, unsigned char \*ciphertext) {

AES\_KEY encrypt\_key;

AES\_set\_encrypt\_key(key, 256, &encrypt\_key);

for (int i = 0; i < len; i += AES\_BLOCK\_SIZE) {

AES\_encrypt(plaintext + i, ciphertext + i, &encrypt\_key);

}

}

void aes\_encrypt\_cbc(const unsigned char \*plaintext, int len, const unsigned char \*key, unsigned char \*ciphertext, unsigned char \*iv) {

AES\_KEY encrypt\_key;

AES\_set\_encrypt\_key(key, 256, &encrypt\_key);

AES\_cbc\_encrypt(plaintext, ciphertext, len, &encrypt\_key, iv, AES\_ENCRYPT);

}

void aes\_encrypt\_cfb(const unsigned char \*plaintext, int len, const unsigned char \*key, unsigned char \*ciphertext, unsigned char \*iv) {

AES\_KEY encrypt\_key;

AES\_set\_encrypt\_key(key, 256, &encrypt\_key);

int num = 0;

AES\_cfb128\_encrypt(plaintext, ciphertext, len, &encrypt\_key, iv, &num, AES\_ENCRYPT);

}

int main() {

unsigned char key[32] = "thisisaverysecurekeyof32bytes!";

unsigned char iv[AES\_BLOCK\_SIZE];

RAND\_bytes(iv, sizeof(iv));

unsigned char plaintext[64] = "Sensitive Data That Needs Encryption";

int len = strlen((char \*)plaintext);

pad\_message(plaintext, &len);

unsigned char ciphertext\_ecb[len];

unsigned char ciphertext\_cbc[len];

unsigned char ciphertext\_cfb[len];

aes\_encrypt\_ecb(plaintext, len, key, ciphertext\_ecb);

aes\_encrypt\_cbc(plaintext, len, key, ciphertext\_cbc, iv);

aes\_encrypt\_cfb(plaintext, len, key, ciphertext\_cfb, iv);

printf("ECB Encrypted: ");

for (int i = 0; i < len; i++) printf("%02x", ciphertext\_ecb[i]);

printf("\n");

printf("CBC Encrypted: ");

for (int i = 0; i < len; i++) printf("%02x", ciphertext\_cbc[i]);

printf("\n");

printf("CFB Encrypted: ");

for (int i = 0; i < len; i++) printf("%02x", ciphertext\_cfb[i]);

printf("\n");

return 0;

}

**4. Write a program for Encrypt and decrypt in cipher block chaining mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES, DES. Test data for S-DES using a binary initialization vector of 1010 1010. A binary plaintext of 0000 0001 0010 0011 encrypted with a binary key of 01111 11101 should give a binary plaintext of 1111 0100 0000 1011. Decryption should work correspondingly.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/des.h>

#include <openssl/rand.h>

// Function to convert binary string to byte array

void bin\_to\_bytes(const char \*bin, unsigned char \*bytes, int len) {

for (int i = 0; i < len; i++) {

bytes[i] = (bin[i \* 8] - '0') << 7 |

(bin[i \* 8 + 1] - '0') << 6 |

(bin[i \* 8 + 2] - '0') << 5 |

(bin[i \* 8 + 3] - '0') << 4 |

(bin[i \* 8 + 4] - '0') << 3 |

(bin[i \* 8 + 5] - '0') << 2 |

(bin[i \* 8 + 6] - '0') << 1 |

(bin[i \* 8 + 7] - '0');

}

}

// S-DES Encryption & Decryption in CBC mode

void sdes\_encrypt\_cbc(const unsigned char \*plaintext, unsigned char \*ciphertext, const unsigned char \*key, unsigned char \*iv) {

DES\_cblock des\_key;

DES\_key\_schedule schedule;

memcpy(des\_key, key, 8);

DES\_set\_key(&des\_key, &schedule);

DES\_ncbc\_encrypt(plaintext, ciphertext, 4, &schedule, (DES\_cblock \*)iv, DES\_ENCRYPT);

}

void sdes\_decrypt\_cbc(const unsigned char \*ciphertext, unsigned char \*plaintext, const unsigned char \*key, unsigned char \*iv) {

DES\_cblock des\_key;

DES\_key\_schedule schedule;

memcpy(des\_key, key, 8);

DES\_set\_key(&des\_key, &schedule);

DES\_ncbc\_encrypt(ciphertext, plaintext, 4, &schedule, (DES\_cblock \*)iv, DES\_DECRYPT);

}

int main() {

const char \*bin\_plaintext = "0000000100100011";

const char \*bin\_key = "0111111101";

const char \*bin\_iv = "10101010";

unsigned char plaintext[2], key[8], iv[8], ciphertext[2], decrypted[2];

bin\_to\_bytes(bin\_plaintext, plaintext, 2);

bin\_to\_bytes(bin\_key, key, 8);

bin\_to\_bytes(bin\_iv, iv, 8);

sdes\_encrypt\_cbc(plaintext, ciphertext, key, iv);

sdes\_decrypt\_cbc(ciphertext, decrypted, key, iv);

printf("Encrypted: ");

for (int i = 0; i < 2; i++) printf("%02X", ciphertext[i]);

printf("\nDecrypted: ");

for (int i = 0; i < 2; i++) printf("%02X", decrypted[i]);

printf("\n");

return 0;

}

**5.Write a program for RSA system, the public key of a given user is e = 31, n = 3599. What is the private key of this user? Hint: First use trial-and-error to determine p and q; then use the extended Euclidean algorithm to find the multiplicative inverse of 31 modulo f(n).**

#include <stdio.h>

// Function to compute gcd

typedef struct {

int gcd, x, y;

} ExtendedGCDResult;

ExtendedGCDResult extended\_gcd(int a, int b) {

if (b == 0)

return (ExtendedGCDResult){a, 1, 0};

ExtendedGCDResult res = extended\_gcd(b, a % b);

return (ExtendedGCDResult){res.gcd, res.y, res.x - (a / b) \* res.y};

}

// Function to compute modular inverse

int mod\_inverse(int e, int phi) {

ExtendedGCDResult res = extended\_gcd(e, phi);

if (res.gcd != 1)

return -1; // No modular inverse exists

return (res.x % phi + phi) % phi;

}

int main() {

int e = 31, n = 3599;

int p = 59, q = 61; // Determined by trial and error

int phi = (p - 1) \* (q - 1);

int d = mod\_inverse(e, phi);

if (d == -1) {

printf("No modular inverse found.\n");

} else {

printf("Private Key (d): %d\n", d);

}

return 0;

}

**6. Write a program for Diffie-Hellman protocol, each participant selects a secret number x and sends the other participant ax mod q for some public number a. What would happen if the participants sent each other xa for some public number a instead? Give at least one method Alice and Bob could use to agree on a key. Can Eve break your system without finding the secret numbers? Can Eve find the secret numbers?**

#include <stdio.h>

#include <math.h>

// Function to compute a^b % mod

long long power(long long a, long long b, long long mod) {

long long result = 1;

a = a % mod;

while (b > 0) {

if (b % 2 == 1) {

result = (result \* a) % mod;

}

a = (a \* a) % mod;

b = b / 2;

}

return result;

}

// Diffie-Hellman Key Exchange

void diffie\_hellman(long long a, long long q, long long x, long long y) {

long long A = power(a, x, q); // Alice's public key

long long B = power(a, y, q); // Bob's public key

// Exchange public keys and calculate shared secret

long long S\_A = power(B, x, q); // Alice's shared secret

long long S\_B = power(A, y, q); // Bob's shared secret

printf("Alice's Public Key: %lld\n", A);

printf("Bob's Public Key: %lld\n", B);

printf("Shared Secret Key: %lld\n", S\_A); // Alice's and Bob's shared key

}

int main() {

long long q = 23; // Prime number

long long a = 5; // Public base

// Secret keys

long long x = 6; // Alice's secret key

long long y = 15; // Bob's secret key

diffie\_hellman(a, q, x, y);

return 0;

}