한승환

- RSA & Hill Cipher 암호화 메시지 전송 프로그램

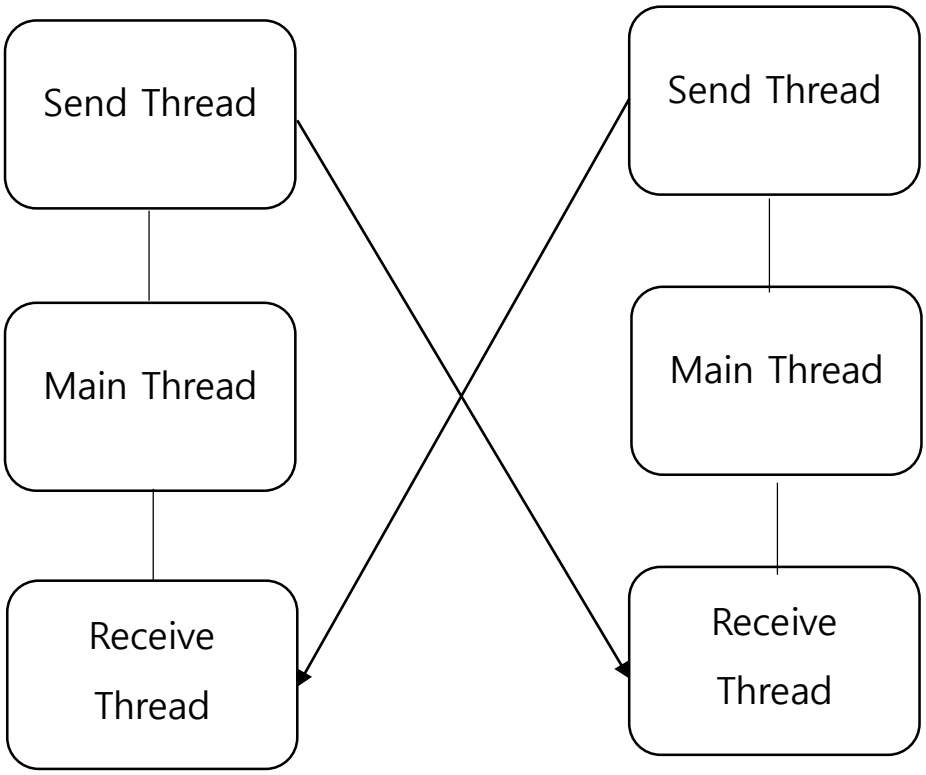
1. 기능

- RSA public, private key로 암호화 된 Hill Cipher 암호화 key를 Client에 보내서 서버와 클라이언트 간 메시지를 Hill Cipher 암호화 key로 암호화하여 기밀성을 유지함

2. 목적

- 메시지의 기밀성 유지 및 암호화 원리 체감
- 수업시간에 배운 내용을 기초로 하여 과제로 진행했음

3. 프로그램 구조



- 작동 순서

1. Client 측에서 RSA private, public 키 생성 -> Server로 RSA public key 보냄
2. Server는 받은 RSA public key로 미리 만들어 둔 Hill Cipher 암호화 키를 암호화하여 Client로 보냄
3. Client는 Server에서 받은 암호화된 Hill Cipher 암호화 키를 자신의 RSB private 키로 복호화한 후, 암호화 키에서 모듈로 연산을 통해서 Hill Cipher 복호화 키를 만듦
4. Server와 Client 간에 서로 암호화, 복호화 하는데 필요한 Hill Cipher 암호화, 복호화 키를 가지고 있으므로 통신이 가능하기 때문에, 이때부터 메시지 전송 시작

4. 기능 구현 원리

- Server & Client 통신

TCP/IP Socket 프로그래밍으로 구현함. Sender thread와 Receiver thread를 생성하여 메시지의 주고받음을 동시에 할 수 있도록 함

```
SOCKET PrepareSocket(char *ipaddr, int port)
{
    SOCKET ConnectSocket = INVALID_SOCKET;

    struct sockaddr_in server;

    printf("\nInitializing Winsock...\n");
    // Initialize Winsock
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR) {
        printf("WSAStartup failed with error: %ld\n", iResult);
        return NULL;
    }
    printf("Initialized\n");

    ConnectSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (ConnectSocket == INVALID_SOCKET)
    {
        printf("socket failed with error : %ld\n", WSAGetLastError());
        WSACleanup();
        return NULL;
    }

    ZeroMemory(&server, sizeof(sockaddr_in));
    server.sin_family = AF_INET;
    inet_pton(AF_INET, ipaddr, (PVOID)&server.sin_addr.s_addr);
    server.sin_port = htons(port);

    if (connect(ConnectSocket, (struct sockaddr *)&server, sizeof(server))<0)
    {
        printf("\n Error : Connect Failed, Error code : %d\n", WSAGetLastError());
        WSACleanup();
        return NULL;
    }
    printf("Connection established\n");
    return ConnectSocket;
}
```

-> 소켓 통신 준비

```
hThread[0] = (HANDLE)_beginthreadex(NULL, 0, ReceiveDataThread, (void *)&ConnectSocket, 0, NULL);
hThread[1] = (HANDLE)_beginthreadex(NULL, 0, SendDataThread, (void *)&ConnectSocket, 0, NULL);
```

-> thread 생성

```
while ((n = _read(0, sendBuf, DEFAULT_BUFLEN - 1)) > 0)
{
    char *encrypted_msg = encrypt(sendBuf, n - 1, size);
    sendBuf[n - 1] = '\0';
    printf("Send : %s -> %s ", sendBuf, encrypted_msg);
    sendResult = send(ConnectSocket, encrypted_msg/*encrypted data*/, strlen(encrypted_msg), 0);
    if (sendResult == SOCKET_ERROR)
    {
        printf("send failed with error : %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        return 1;
    }
    free(encrypted_msg);
    if (strncmp("quit", sendBuf, 4) == 0)
    {
        sendResult = shutdown(ConnectSocket, SD_BOTH);
        if (sendResult == SOCKET_ERROR)
        {
            printf("shutdown failed with error : %d\n", WSAGetLastError());
            closesocket(ConnectSocket);
            return 1;
        }
        break;
    }
    printf("(%d bytes send)\n", sendResult);
}
closesocket(ConnectSocket);
return 0;
```

-> Sender thread

```
do{
    ZeroMemory(recvBuf, sizeof(recvBuf));
    recvResult = recv(ConnectSocket, recvBuf, recvbuflen, 0);
    if (recvResult > 0)
    {
        char *decrypted_msg = decrypt(recvBuf, recvResult, size);

        if (strncmp("quit", decrypted_msg, 4) == 0)
        {
            printf("Connection closing...\n");
            free(decrypted_msg);
            break;
        }
        // decyprt & print message
        printf("Received : %s -> %s\n", recvBuf, decrypted_msg);
        free(decrypted_msg);
    }
    else if (recvResult == 0)
        printf("Connection closing...\n");
    else
    {
        int wsaError = WSAGetLastError();
        if (wsaError == WSAEINTR)
            printf("\nConnection closing...\n");
        else
            printf("\nrecv failed with error : %d\n", wsaError);
        closesocket(ConnectSocket);
        TerminateThread(hThread[1], recvResult);
        return 1;
    }
} while (recvResult > 0);

recvResult = shutdown(ConnectSocket, SD_BOTH);
if (recvResult == SOCKET_ERROR)
{
    printf("shutdown failed with error : %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    return 1;
}

closesocket(ConnectSocket);
TerminateThread(hThread[1], recvResult);
return 0;
```

-> Receiver thread

- RSA 키 쌍 생성

```
unsigned int p, q, _N, e;
while (1)
{
    srand(time(NULL));
    p = rand();
    if (CheckPrime(p))
    break;
}
while (1)
{
    srand(time(NULL));
    q = rand();
    if (CheckPrime(q))
    break;
}
while (1)
{
    srand(time(NULL));
    e = rand() % _N;
    if (_N % e != 0)
    break;
}
N = p * q;
_N = (p - 1) * (q - 1);
sprintf_s(sendBuf, sizeof(sendBuf), "%d %d", N, e);
int initBufLength = strlen(sendBuf);
sendBuf[initBufLength] = '\0';

send(ConnectSocket, sendBuf, initBufLength + 1, 0);
```

-> RSA public key 서버에게 보냄

- Hill Cipher 암호화, 복호화 키 생성

```
// dynamically allocate
key = MakeNewMatrix(size);

int detValue = 0;
int mod_detValue = 0;
//int inverse_detValue = 0;
int value = 0;
while (1)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            int temp = 0;
            //scanf_s("%d", &temp);
            //key[i][j] = temp % 31;
            key[i][j] = rand() % 31;
        }
    }

    if ((detValue = determinant(key, size) % 31) != 0) // if invertible matrix (mod 31)
    {
        mod_detValue = CalculateModulo(detValue, 31);   // always positive value
        inverse_detValue = ModMulInv(mod_detValue, 31);
        break;
    }
    printf("Cannot use\n");
}
```

-> 암호화 키 생성

```c
int determinant(int **matrix, int size)
{
    if (size == 1)
        return matrix[0][0];

    int index = 0;
    int detValue = 0;
    for (index = 0; index < size; index++)
    {
        // make a minor matrix
        int **minor_matrix = MakeNewMatrix(size - 1);
        MakeMinorMatrix(matrix, minor_matrix, 0, index, size - 1);

        // Caculate determinant value of each partial matrix
        detValue += matrix[0][index] * SIGNUNSIGN(0 + 1, index + 1) * determinant(minor_matrix, size - 1);

        // free allocated memory
        free(minor_matrix);
    }
    return detValue;
}
```

```c
int **MakeInverseMatrix(int **matrix, int ModInv_detValue, int size)
{
    // inverse of matrix = Modular multipication inverse of |matrix|mod31 * [cofactor of (i,j)]^T

    // make a new matrix for inverse
    int **new_matrix_inv = MakeNewMatrix(size);

    // Make an adjoint matrix
    MakeAdjointMatrix(matrix, new_matrix_inv, size);

    // Multiply the determinant value of the matrix to the adjoint matrix
    MultiplyIntegerToMatrixMod31(ModInv_detValue, new_matrix_inv, size);
    return new_matrix_inv;
}

void MakeAdjointMatrix(int **matrix, int **adjoint_matrix, int size)
{
    for (int i = 0; i < size; i++)
    {
        int cofactor = 0;
        for (int j = 0; j < size; j++)
        {
            cofactor = 0;
            int **minor_matrix = MakeNewMatrix(size - 1);
            MakeMinorMatrix(matrix, minor_matrix, i, j, size - 1);

            cofactor += SIGNUNSIGN(i + 1, j + 1) * determinant(minor_matrix, size - 1);
            adjoint_matrix[j][i] = cofactor;
            free(minor_matrix);
        }
    }
}

void MultiplyIntegerToMatrixMod31(int value, int **matrix, int size)
{
    for (int row = 0; row < size; row++)
        for (int col = 0; col < size; col++)
            matrix[row][col] = CalculateModulo(matrix[row][col] * value, 31);
}
```

-> Hill Cipher 복호화 키 생성

- Hill Chiper 암호화 key를 RSA public로 암호화하기

```
unsigned int RSAEncrypt(unsigned int plain)
{
    return RepeatedSquareMod(plain);
}

unsigned int RepeatedSquareMod(unsigned int base)
{
    if (base == 1) return 1;
    int exp_memory_size = 0;
    for (exp_memory_size = 0; exp_memory_size < 32 && e >= (1 << exp_memory_size); exp_memory_size++);

    unsigned int *exp_mod_memory = (unsigned int *)malloc(sizeof(unsigned int) * exp_memory_size);
    exp_mod_memory[0] = CalculateModulo(base, N);
    int prev, next;
    for (int i = 0; i < exp_memory_size - 1; i++)
    {
        prev = e >> exp_memory_size - (i + 1);
        next = e >> exp_memory_size - (i + 2);
        exp_mod_memory[i + 1] = CalculateModulo(exp_mod_memory[i] * exp_mod_memory[i], N);
        if ((prev << 1) != next)
            exp_mod_memory[i + 1] = CalculateModulo(exp_mod_memory[i + 1] * exp_mod_memory[0], N);
    }
    unsigned int result = exp_mod_memory[exp_memory_size - 1];
    free(exp_mod_memory);
    return result;
}
```

```
printf("Sending the hill cihper key to client with RSA public key\n");
unsigned int encrypted_size = RSAEncrypt(size);
sprintf_s(sendBuf, sizeof(sendBuf) - 1, "%u ", encrypted_size);
int num_size = strlen(sendBuf);
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        char char_num[20]; ZeroMemory(char_num, sizeof(char_num));
        sprintf_s(char_num, sizeof(char_num) - 1, "%u ", RSAEncrypt(key[i][j]));
        strncpy(sendBuf + num_size, char_num, strlen(char_num));
        num_size += strlen(char_num);
    }
}
```

```
// send the hill cipher key to client
sendBuf[DEFAULT_BUFLEN - 1] = '\0';
sendResult = send(clientSocket, sendBuf/*encrypted data*/, DEFAULT_BUFLEN, 0);
if (sendResult == SOCKET_ERROR)
{
    printf("send failed with error : %d\n", WSAGetLastError());
    closesocket(clientSocket);
    return 1;
}
printf("Key is sent with RSA client's public key\n");
```

- Hill Cipher 로 암호화, 복호화

```
char *encrypted_msg = encrypt(sendBuf, n - 1, size);
sendBuf[n - 1] = '\0';
printf("Send : %s -> %s ", sendBuf, encrypted_msg);
sendResult = send(ConnectSocket, encrypted_msg/*encrypted data*/, strlen(encrypted_msg), 0);
```

```
char *encrypt(_In_ char block[], _In_ int block_len, _In_ int size)
{
    int numberOfBlock = ((block_len % size > 0) ? block_len / size + 1 : block_len / size) * size + 1;
    char *encrypted_block = (char *)malloc(sizeof(char) * numberOfBlock);
    ZeroMemory(encrypted_block, sizeof(char) * numberOfBlock);
    int count = 0;
    int *subBlock = NULL;
    while (count + size <= block_len)
    {
        subBlock = MultiplyMatrixToKey(&block[count], size);
        for (int idx = 0; idx < size; idx++)
            encrypted_block[count++] = CONVERT_TO_CHAR(subBlock[idx]);
        free(subBlock);
    }

    if (count < block_len)
    {
        int remainder = block_len - count;
        char *remainder_block = (char *)malloc(sizeof(char) * size);
        for (int i = 0; i < size; i++)
        {
            if (i < remainder)
                remainder_block[i] = block[count + i];
            else
                remainder_block[i] = '_';
        }

        int *subBlock = MultiplyMatrixToKey(remainder_block, size);
        for (int idx = 0; idx < size; idx++)
            encrypted_block[count + idx] = CONVERT_TO_CHAR(subBlock[idx]);
        free(remainder_block);
    }
    encrypted_block[numberOfBlock - 1] = '\0';
    return encrypted_block;
}
```

-> 암호화

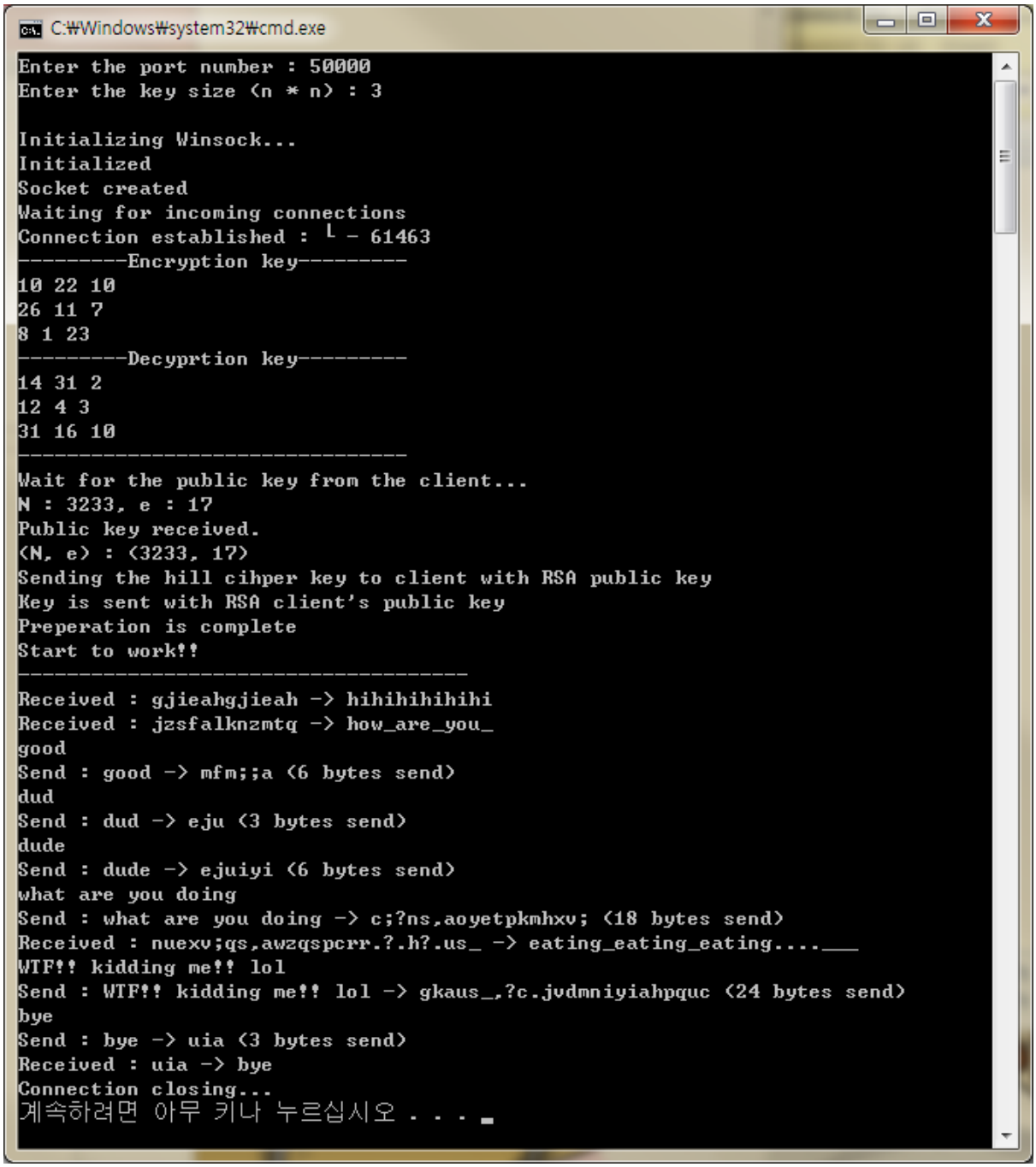
```
ZeroMemory(recvBuf, sizeof(recvBuf));
recvResult = recv(clientSocket, recvBuf, recvbuflen, 0);
if (recvResult > 0)
{
    char *decrypted_msg = decrypt(recvBuf, recvResult, size);

    if (strncmp("quit", decrypted_msg, 4) == 0)
    {
        printf("Connection closing...\n");
        break;
    }

    // decyprt & print message
    printf("Received : %s -> %s\n", recvBuf, decrypted_msg);
}
```

```
char *decrypt(_In_ char block[], _In_ int block_len, _In_ int size)
{
    char *decrypted_msg = (char *)malloc(sizeof(char) * block_len + 1);
    ZeroMemory(decrypted_msg, sizeof(char) * block_len + 1);
    int count = 0;
    int *subBlock = NULL;
    while (count + size <= block_len)
    {
        subBlock = MultiplyMatrixToInvKey(&block[count], size);
        for (int idx = 0; idx < size; idx++)
            decrypted_msg[count++] = CONVERT_TO_CHAR(subBlock[idx]);
        free(subBlock);
    }
    decrypted_msg[count] = '\0';
    return decrypted_msg;
}
```

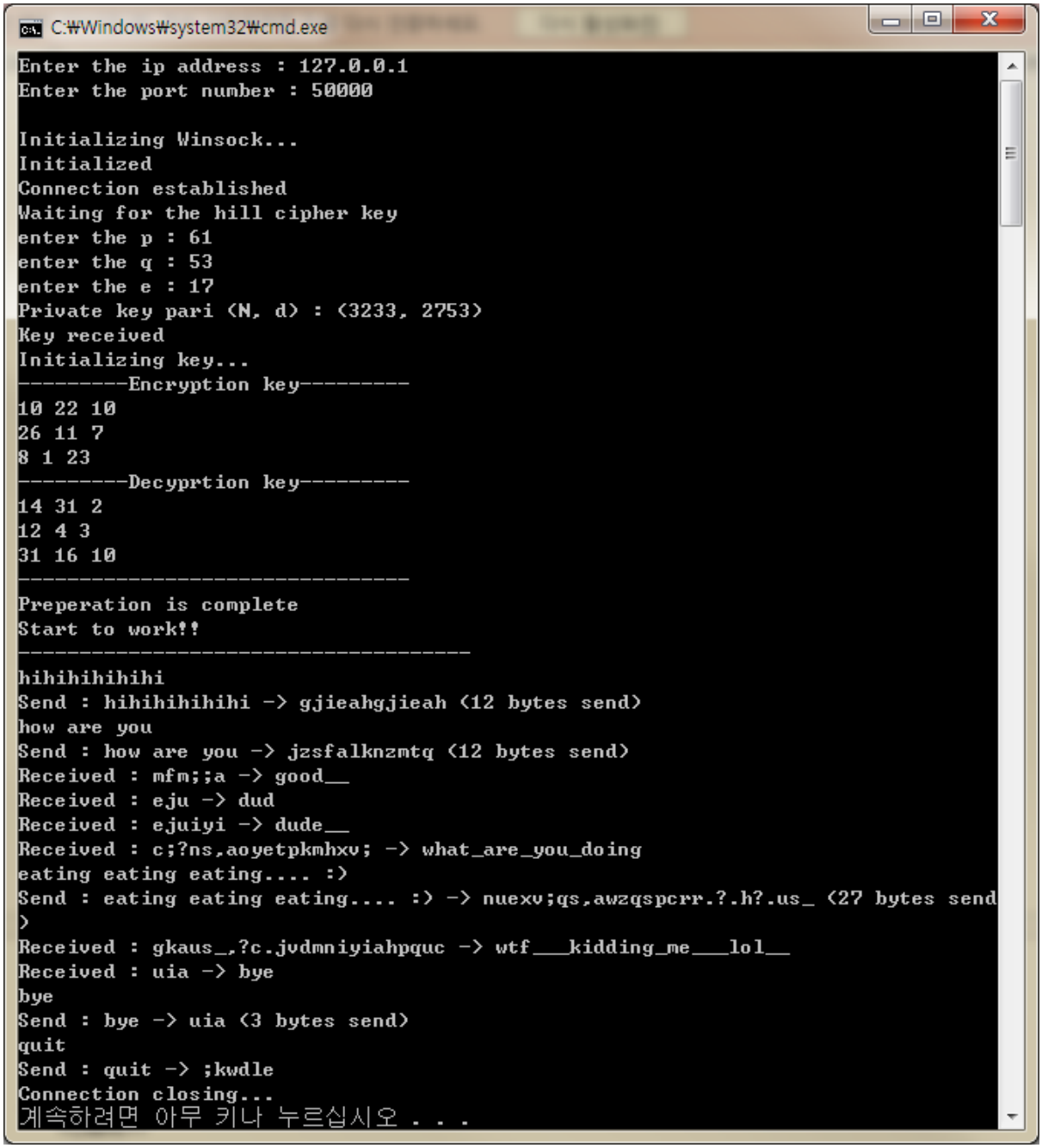
-> 복호화

5. 프로그램 실행 모습

<Server>



<Client>


C:₩Windows₩system32₩cmd.exe
Enter the ip address : 127.0.0.1
Enter the port number : 50000
Initializing Winsock...
Initialized
Connection established
Waiting for the hill cipher key
enter the p : 61
enter the q : 53
enter the e : 17
Private key pari (N, d) : (3233, 2753)
Key received
Initializing key...
---------Encryption key---------
10 22 10
26 11 7
8 1 23
---------Decyprtion key---------
14 31 2
12 4 3
31 16 10

Preperation is complete
Start to work!!
----
hihihihihihi
Send : hihihihihihi -> gjieahgjieah (12 bytes send)
how are you
Send : how are you -> jzsfalknzmtq (12 bytes send)
Received : mfm;;a -> good__
Received : eju -> dud
Received : ejuiyi -> dude__
Received : c;?ns,aoyetpkmhxv; -> what_are_you_doing
eating eating eating.... :)
Send : eating eating eating.... :) -> nuexv;qs,awzqspcrr.?.h?.us_ (27 bytes send
)
Received : gkaus_,?c.jvdmniyiahpquc -> wtf___kidding_me___lol__
Received : uia -> bye
bye
Send : bye -> uia (3 bytes send)
quit
Send : quit -> ;kwdle
Connection closing...
계속하려면 아무 키나 누르십시오 . . .