Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

**Лабораторная работа №3**

«Применение потоковых шифров НС-128, Rabbit, Salsa20 и Sosemanuk»

Выполнил:

студент гр. МГ-211 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Бурдуковский И.А./

подпись

Проверил:

Профессор

кафедры ПМиК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фионов А.Н./

Новосибирск

2023 г.

**ОГЛАВЛЕНИЕ**

[Задание 3](#_Toc135774551)

[Выполнение 4](#_Toc135774552)

[Листинг 5](#_Toc135774553)

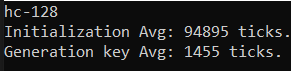
# Задание

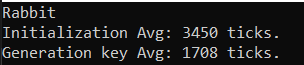
1. Для всех четырёх шифров (НС-128, Rabbit, Salsa20 и Sosemanuk) сравнить длительность функций инициализации, и время генерации ключевого потока.
2. Выбрать один понравившийся потоковый шифр и реализовать на его основе приложение для передачи файла в зашифрованном виде по сети. Сравнить с соответствующей реализацией на основе блочного шифра.

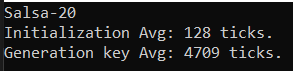
# Выполнение

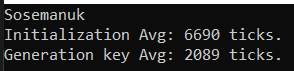
1. Реализованы необходимые алгоритмы. Для сравнения было решено взять результаты за 256 повторений. Результирующее время в тиках. Рассчитывается как среднее арифметическое.

Результаты:









Подводя итоги, можно сказать, что по результатам в инициализации лидирует – Salsa20 с заметным отрывом. По времени кодирования – HC-128, но Rabbit очень близок к результатам HC-128.

1. Для преобразования в клиент-серверное приложение был взят алгоритм HC-128.

Структуру сообщения которая будет включать шифротекст и вектор инициализации:

Сервер отдает и принимает пакеты для каждого клиента в потоке. Клиент устанавливает вектор инициализации и шифрует сообщение которое потом кладется в поле message. Другой клиент получает пакет и также устанавливает вектор инициализации и декодирует сообщение.

# Листинг

struct myMessageStruct

{

char username[32];

u8 iv[16];

u8 message[64];

char pad\_0[144];

};

std::string GetLastErrorAsString()

{

//Get the error message, if any.

DWORD errorMessageID = ::GetLastError();

if (errorMessageID == 0)

return std::string(); //No error message has been recorded

LPSTR messageBuffer = nullptr;

size\_t size = FormatMessageA(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, errorMessageID, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPSTR)&messageBuffer, 0, NULL);

std::string message(messageBuffer, size);

//Free the buffer.

LocalFree(messageBuffer);

return message;

}

std::string WSAGetLastErrorAsString()

{

//Get the error message, if any.

DWORD errorMessageID = ::WSAGetLastError();

if (errorMessageID == 0)

return std::string(); //No error message has been recorded

LPSTR messageBuffer = nullptr;

size\_t size = FormatMessageA(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, errorMessageID, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPSTR)&messageBuffer, 0, NULL);

std::string message(messageBuffer, size);

//Free the buffer.

LocalFree(messageBuffer);

return message;

}

void ClearScreen()

{

HANDLE hStdOut;

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

DWORD count;

DWORD cellCount;

COORD homeCoords = { 0, 0 };

hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hStdOut == INVALID\_HANDLE\_VALUE) return;

/\* Get the number of cells in the current buffer \*/

if (!GetConsoleScreenBufferInfo(hStdOut, &csbi)) return;

cellCount = csbi.dwSize.X \* csbi.dwSize.Y;

/\* Fill the entire buffer with spaces \*/

if (!FillConsoleOutputCharacter(

hStdOut,

(TCHAR)' ',

cellCount,

homeCoords,

&count

)) return;

/\* Fill the entire buffer with the current colors and attributes \*/

if (!FillConsoleOutputAttribute(

hStdOut,

csbi.wAttributes,

cellCount,

homeCoords,

&count

)) return;

/\* Move the cursor home \*/

SetConsoleCursorPosition(hStdOut, homeCoords);

}

int main(int argc, char\*\* argv)

{

WSADATA wsaData;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo\* result = nullptr,

\* ptr = nullptr,

hints{};

char receive\_buffer[\_chatPacketSize];

// Initialize Winsock

int i\_result = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (i\_result != 0) {

printf("WSAStartup failed with error: %d\n", i\_result);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Resolve the server address and port

i\_result = getaddrinfo("localhost", \_chatPort, &hints, &result);

if (i\_result != 0) {

printf("getaddrinfo failed with error: %d\n", i\_result);

WSACleanup();

return 1;

}

// Attempt to connect to an address until one succeeds

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

// Create a SOCKET for connecting to server

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

return 1;

}

// Connect to server.

i\_result = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (i\_result == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

printf("Unable to connect to server!\n");

WSACleanup();

return 1;

}

std::string s\_name;

std::cout << "Please enter your name: ";

std::getline(std::cin, s\_name);

ECRYPT\_init();

ECRYPT\_keysetup(&ctx, &(key[0]), 128, 128);

while (true)

{

// Receive chat data from server

i\_result = recv(ConnectSocket, receive\_buffer, \_chatPacketSize, 0);

if (i\_result > 0)

{

ClearScreen();

const auto chat\_history = reinterpret\_cast<myMessageStruct\*>(receive\_buffer);

for (int i = \_chatHistoryLength - 1; i > -1; i--)

{

ECRYPT\_ivsetup(&ctx, &(chat\_history[i].iv[0]));

u8 data\_out[64];

ECRYPT\_encrypt\_bytes(&ctx, &(chat\_history[i].message[0]), &data\_out[0], 64);

std::cout << std::setw(sizeof(chat\_history[i].username)) << chat\_history[i].username << " | " <<

(char\*)data\_out << std::endl;

}

}

else if (i\_result == 0)

{

std::cout << "[i] Connection closed!" << std::endl;

}

else

{

std::cout << "[!] (Receive) failed with error: " << WSAGetLastErrorAsString() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

std::string s\_message;

std::cout << "> ";

std::getline(std::cin, s\_message);

if (s\_message == "!quit")

{

std::cout << "[i] Closing connection!" << std::endl;

break;

}

myMessageStruct packet{};

s\_name.resize(sizeof(packet.username), '\0');

packet.username[0] = 0;

s\_name.copy(packet.username, s\_name.size());

packet.username[sizeof(packet.username) - 1] = '\0';

memcpy(packet.iv, iv, 16);

u8 data\_in\_m[64];

data\_in\_m[0] = 0;

s\_message.resize(sizeof(packet.message), '\0');

memcpy(data\_in\_m, s\_message.c\_str(), s\_message.size());

ECRYPT\_ivsetup(&ctx, &(iv[0]));

ECRYPT\_encrypt\_bytes(&ctx, &data\_in\_m[0], &packet.message[0], 64);

/\*s\_message.resize(sizeof(packet.message), '\0');

packet.message[0] = 0;

s\_message.copy(packet.message, s\_message.size());

packet.message[sizeof(packet.message) - 1] = '\0';\*/

// Send message to the server

i\_result = send(ConnectSocket, reinterpret\_cast<char\*>(&packet), \_chatMessageSize, 0);

if (i\_result == SOCKET\_ERROR) {

std::cout << "[!] (Send) failed with error: " << WSAGetLastErrorAsString() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

}

// shutdown the connection since no more data will be sent

i\_result = shutdown(ConnectSocket, SD\_SEND);

if (i\_result == SOCKET\_ERROR) {

std::cout << "[!] (Shutdown) failed with error: " << WSAGetLastErrorAsString() << std::endl;

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

closesocket(ConnectSocket);

WSACleanup();

return 0;

}