БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Практическая работа № 2

по дисциплине «Методы защиты информации»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2017

### ИПР №2

#### Часть 2.1. Алгоритм шифрования RSA

**Цель:** реализовать на языке программирования C++ алгоритм асимметричного шифрования RSA**.**

**Результат:** Программа шифрования, осуществляющая криптографическое преобразование введенного текста с помощью алгоритма симметричного шифрования RSA.

**Общая постановка задачи:**  Создать программу, читающую данные из файла, шифрующие и дешифрующие их с помощью алгоритма RSA.

Порядок использования программы:



Необходимо указать файл для чтения с помощью флага **–i**, выходной файл указывать необязательно, программа автоматически создаст его, но если это необходимо, можно указать после флага **–o**. По умолчанию программа работает в режиме шифрования, для дешифрования необходима указать флаг **–d**

В случае с RSA можно вызвать программу генерации ключей указав флаг **–g**

Стоит отметить, что данная реализация RSA использует слабые ключи размером до 64бит. При использовании RSA также необходимо указать пару значений –**n** и **–m**. Которые будут использованы для шифрования; Ниже будет приведен пример использования программы.



Используя ключи (3,25777) и (16971,25777) зашифровал и расшифровал сообщение



Исходные файлы:

rsa.cpp

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/21/17. //  //crypt = pow(msg,e)%n; //msg = pow(crypt,d)%n = pow(pow(msg,e),d)%n = pow(msg,e\*d)%n  #include <cstdint> #include <cstdlib> #include <ctime> #include <cstdio> #include "rsa.h" //n is known //e for sender [en] - openkey //d for receiver [de] - closedkey //limits: //pow(msh,e\*d)==msg % n | msg < n  **bool** isPrime(uint64\_t number) {  **for** (**int** i = 2; i < number / 2; i++)  {  **if** (number % i == 0) **return false**;  }  **return true**; }  //compute a^b mod c //https://www.topcoder.com/community/data-science/data-science-tutorials/primality-testing-non-deterministic-algorithms/ uint64\_t powm(uint32\_t base,int32\_t exp, uint32\_t modulus){  base %= modulus;  uint64\_t result = 1;  **while** (exp > 0) {  **if** (exp & 1) result = (result \* base) % modulus;  base = (base \* base) % modulus;  exp >>= 1;  }  **return** result; }  uint64\_t gcd(uint64\_t a, uint64\_t b) {  **return** b == 0 ? a : gcd(b, a % b); }  //for our example we'll use max of 8 bits **void** rsa\_generate\_keys(uint32\_t bits, uint32\_t e) {  srand((**unsigned int**) time(0));  //generate random P and Q  uint64\_t p = 0;  uint64\_t q = 0;  // printf("\ne %u",e);   **do**{  p = (uint16\_t) rand();  }**while**(!(isPrime(p) && gcd(e,p-1)==1)); // printf("\np %llu",p);   **do**{  q = (uint16\_t ) rand();  }**while**(!(isPrime(q) && gcd(e,q-1)==1)); // printf("\nq %llu",q);   uint64\_t n = p\*q;  uint64\_t phi = (p-1)\*(q-1);  uint64\_t d = (2\*phi+1)/e;  // printf("\nn %llu",n); // printf("\nphi %llu",phi); // printf("\nd %llu",d);   printf("\npublic key: (n: %llu e: %u)\n",n,e);  printf("\nprivate key: (n: %llu d: %llu)\n",n,d); } |

rsa.h

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/21/17. //  #ifndef IPR1\_RSA\_H #define IPR1\_RSA\_H uint64\_t powm(uint32\_t base,int32\_t exp, uint32\_t modulus); **void** rsa\_generate\_keys(uint32\_t bits, uint32\_t e); **template** <**typename** T> uint32\_t rsa\_process\_block(uint32\_t n,uint32\_t m,T block){  **return** powm(block, m, n); } #endif //IPR1\_RSA\_H |

main.cpp

|  |
| --- |
| #include <cstdio> #include <cstdlib> #include <cstring> #include "des.h" #include "gost.h" #include "rsa.h"  **struct** IPRConfig{  **const char** \*path = "";  **bool** encrypt = **true**;  **bool** genkeys = **false**;  **int** type = 0;//0-DES, 1-28147, 2-RSA  **const char** \*inputFilePath = "";  **const char** \*outputFilePath = "";  **const char** \*key = "";  uint32\_t rsa\_n = 0;  uint32\_t rsa\_m = 0;  **bool** isValid; };  **void** help(){  printf("Usage:\n");  printf("'des' or '28147' or 'rsa' to specify encryption type\n");  printf("'-g': generate keys\n");  printf("'-d': decrypt if no-flag: encrypt\n");  printf("'-o': + output file path\n");  printf("'-i': + input file path\n"); // printf("'-k': + key file path\n"); }  **void** test(){  uint8\_t t = 15;  printf("\nencrypt");  uint32\_t res = rsa\_process\_block(1364143303,3,t);  printf("\ndecrypt %llu",res);  t = (uint8\_t) rsa\_process\_block(1364143303, 909377019, res);  printf("\n t is %d\n",t); }   IPRConfig parseArgs(**char**\* args[], **int** argc){  IPRConfig config;  **if**(argc<2) {  printf("No arguments specified\n");  help();  }**else** {  config.path = args[0];  **int** i;  **for** (i = 1; i < argc; i++) {  printf("Parsing arg %d : %s \n", i, args[i]);  **if** (strcmp(args[i], "des") == 0 || strcmp(args[i], "DES") == 0) {  config.type = 0;  } **else if** (strcmp(args[i], "28147") == 0) {  config.type = 1;  } **else if** (strcmp(args[i], "rsa") == 0 || strcmp(args[i], "RSA") == 0 ) {  config.type = 2;  } **else if** (strcmp(args[i], "o") == 0 || strcmp(args[i], "-o") == 0) {  i++;  **if** (i < argc) {  config.outputFilePath = args[i];  }  } **else if** (strcmp(args[i], "i") == 0 || strcmp(args[i], "-i") == 0) {  i++;  **if** (i < argc) {  config.inputFilePath = args[i];  }  } **else if** (strcmp(args[i], "d") == 0 || strcmp(args[i], "-d") == 0) {  config.encrypt = **false**;  } **else if** (strcmp(args[i], "g") == 0 || strcmp(args[i], "-g") == 0) {  config.genkeys = **true**;  } **else if** (strcmp(args[i], "k") == 0 || strcmp(args[i], "-k") == 0) {  i++;  **if** (i < argc) {  config.key = args[i];  }  } **else if** (strcmp(args[i], "n") == 0 || strcmp(args[i], "-n") == 0) {  i++;  **if** (i < argc) {  config.rsa\_n = (uint32\_t) atol(args[i]);  }  } **else if** (strcmp(args[i], "m") == 0 || strcmp(args[i], "-m") == 0) {  i++;  **if** (i < argc) {  config.rsa\_m = (uint32\_t) atol(args[i]);  }  } **else** {  printf("\nWrong arg %d : %s . Will be ignored\n", i, args[i]);  }  }  }   //check config validity  config.isValid = config.genkeys || (strcmp(config.inputFilePath,"") != 0);  **if**(config.isValid && strcmp(config.outputFilePath, "") == 0) {  **auto** outPath = (**char** \*) malloc(**sizeof**(config.inputFilePath) + **sizeof**("\_decrypted"));  strcpy(outPath,config.inputFilePath);  strcat(outPath,config.encrypt?"\_encrypted":"\_decrypted");  config.outputFilePath = outPath;  }   **return** config; }  **int** main(**int** argc, **char**\* argv[]) {  **auto** config = parseArgs(argv,argc);  **if**(config.isValid){   **if**(config.genkeys){  rsa\_generate\_keys(8,3);  }**else** {   FILE \*inputFile = fopen(config.inputFilePath, "rb");  **if** (!inputFile) {  printf("\ninputFile opening failed: %s\n", config.inputFilePath);  **return** EXIT\_FAILURE;  }   FILE \*outputFile = fopen(config.outputFilePath, "wb");  **if** (!outputFile) {  printf("\noutputFile opening failed: %s\n", config.outputFilePath);  **return** EXIT\_FAILURE;  }    **if** (config.type == 2) {  printf("%s %s N:%u M:%u\n", "RSA", config.encrypt ? "encrypt" : "decrypt",config.rsa\_n,config.rsa\_m);  uint8\_t buf8 = 0;  uint32\_t buf32 = 0;  //check  **if**(config.encrypt) {  **while** (fread(&buf8, **sizeof**(buf8), 1, inputFile) == 1) {  buf32 = rsa\_process\_block(config.rsa\_n, config.rsa\_m, buf8);  fwrite(&buf32, **sizeof**(buf32), 1, outputFile);  }  }**else**{  **while** (fread(&buf32, **sizeof**(buf32), 1, inputFile) == 1) {  buf8 = (uint8\_t) rsa\_process\_block(config.rsa\_n, config.rsa\_m, buf32);  fwrite(&buf8, **sizeof**(buf8), 1, outputFile);  }  }  } **else** {  printf("Gen keys\n");  //gostKeys;  uint32\_t keysG[8] = {0x33206D54, 0x326C6568, 0x20657369, 0x626E7373, 0x79676120, 0x74746769, 0x65686573,  0x733D2C20};  //desKeys;  uint64\_t \*keys = **new** uint64\_t[16];  uint64\_t initKey = 1383827165325090801;  des\_generate\_keys(keys, initKey);  printf("Keys done\n");   printf("%s %s\n", config.type == 0 ? "des" : "gost28147", config.encrypt ? "encrypt" : "decrypt");  uint64\_t buf = 0;  **while** (fread(&buf, **sizeof**(buf), 1, inputFile) == 1) {  **switch** (config.type) {  **case** 0:  des\_process\_block(&buf, keys, config.encrypt);  **break**;  **case** 1:  gost28147\_process\_block(&buf, keysG, config.encrypt);  **break**;  **default**:  **break**;  }  fwrite(&buf, **sizeof**(buf), 1, outputFile);  }  }  fclose(inputFile);  fclose(outputFile);  }  }  **return** 0; } |