БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Практическая работа № 1

по дисциплине «Методы защиты информации»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2017

### ИПР №1

#### Часть 1.1 Алгоритмы шифрования DES и ГОСТ

**Цель:** реализовать на языке программирования C++ алгоритмы симметричного шифрования DES и ГОСТ 28147**.**

**Результат:** Программа шифрования, осуществляющая криптографическое преобразование введенного текста с помощью алгоритмов симметричного шифрования DES и ГОСТ 28147.

**Общая постановка задачи:**  Создать программу, читающую данные из файла и шифрующие их с помощью алгоритмов DES и ГОСТ 28147.

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**

Алгоритм **DES** использует указанные в стандарте s-boxes, перестановки. Они находятся в заголовочном файле **des.h**. Реализация функций в **des.cpp**.

Генерация ключей и проверка предусмотрена, но в примере используется конкретный ключ, его можно изменить в исходнике **main.cpp**

Алгоритм ГОСТ 28147 использует s-boxes из документа **RFC 4357.** Генерация ключейне предусмотрена, используется пользовательский ключ, аналогично можно заменить в исходнике **main.cpp.** Константы и доступныеметоды указаныв файле **gost.h**, реализация в – **gost.cpp**

Ниже приведен пример использования программы:



В результате был создан новый файл test.txt\_encrypted:



Попробуем расшифровать используя флаг –d



В результате работы был создан новый файл test.txt\_encrypted\_decrypted:



Аналогично протестируем работу алгоритма ГОСТ 28147



Исходные файлы:

Utility.h

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/21/17. //  #ifndef IPR1\_UTILITY\_H #define IPR1\_UTILITY\_H  #include <cstdio> #include <cstdint>  **template** <**class** T,**class** U> **void** set\_bit(T\* target,**const** U src,**const int** from,**const int** to){  \*target ^= (-((src & (1ULL << from)) != 0) ^ \*target) & (1ULL << to); }  **template** <**class** T> **void** cycle\_shift\_left(T\* target, **const** T src,**const int** size, **int** count){  **for**(**int** i = 0;i<count;i++) {  set\_bit(target, src, size - count + i, i);  }  **for**(**int** i = 0;i<size-count;i++){  set\_bit(target,src,i,i+count);  } }  **template** <**class** T,**class** U> **void** permutation(T \*target, **const** U src, **const int**\* map, **int** length){  **for**(**int** i = 0; i<length; i++){  set\_bit(target,src,map[i]-1,i);  } }  **template** <**class** T,**class** U> **void** substitution(T \*target, **const** U src, **const int** blockSize, **const int** newBlockSize, **const int** blockCount, **const int** columns, **const int** \*map,**bool** linear = **false**){  **for**(**int** i = 0; i < blockCount; i++)  {  **int** subblock = 0;  **int** substituted = 0;  //split block  **for**(**int** j = 0; j<blockSize; j++) {  set\_bit(&subblock, src, i \* blockSize + j, j);  }   **if**(linear){  substituted = map[i\*columns+subblock];  }**else** {  uint32\_t y = 0;  uint32\_t x = 0;  //get y  set\_bit(&y, subblock, 0, 0);  set\_bit(&y, subblock, blockSize - 1, 1);  //get x  **for** (**int** j = 1, k = 0; j < (blockSize - 2); j++, k++) {  set\_bit(&x, subblock, j, k);  }  substituted = map[i\*64+y\*columns+x];  }   //join block  **for**(**int** j = 0; j<newBlockSize; j++){  set\_bit(target, substituted, j,i\*newBlockSize + j);  }  } }  **template** <**class** T> **void** print\_bits(T x,**int** size = 0){  **int** numbits = size == 0 ? (8\***sizeof**(T)) : size;  **while**(--numbits >= 0) printf(numbits%8 == 0 ? "%c ":"%c", (x & ((T)1<<numbits)) ? '1' : '0');  printf("\n"); }  #endif //IPR1\_UTILITY\_H |

des.h

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/17/17. //  #ifndef IPR1\_DES\_H #define IPR1\_DES\_H #include <cstdint> #include <cstdio> #include "utility.h" **void** des\_generate\_keys(uint64\_t\* keys,**const** uint64\_t startKey); **void** des\_expand(uint64\_t \*target,**const** uint32\_t src); uint32\_t des\_f\_round(**const** uint32\_t data, uint64\_t key); **void** des\_process\_block(uint64\_t\* block, uint64\_t\* keys,**bool** encrypt); **void** des\_decrypt\_feistel(uint32\_t \*left, uint32\_t \*right, **int** rounds, uint64\_t \*key); **void** des\_encrypt\_feistel(uint32\_t\* left, uint32\_t\* right, **int** rounds, uint64\_t\* key);   **const int** des\_ip[64] = {  58, 50, 42, 34, 26, 18, 10, 2,  60, 52, 44, 36, 28, 20, 12, 4,  62, 54, 46, 38, 30, 22, 14, 6,  64, 56, 48, 40, 32, 24, 16, 8,  57, 49, 41, 33, 25, 17, 9, 1,  59, 51, 43, 35, 27, 19, 11, 3,  61, 53, 45, 37, 29, 21, 13, 5,  63, 55, 47, 39, 31, 23, 15, 7};  **const int** des\_fp[64] = {  40, 8, 48, 16, 56, 24, 64, 32,  39, 7, 47, 15, 55, 23, 63, 31,  38, 6, 46, 14, 54, 22, 62, 30,  37, 5, 45, 13, 53, 21, 61, 29,  36, 4, 44, 12, 52, 20, 60, 28,  35, 3, 43, 11, 51, 19, 59, 27,  34, 2, 42, 10, 50, 18, 58, 26,  33, 1, 41, 9, 49, 17, 57, 25};   **const int** des\_key\_permutation[104] = { //1 56  57, 49, 41, 33, 25, 17, 9,  1, 58, 50, 42, 34, 26, 18,  10, 2, 59, 51, 43, 35, 27,  19, 11, 3, 60, 52, 44, 36,  63, 55, 47, 39, 31, 23, 15,  7, 62, 54, 46, 38, 30, 22,  14, 6, 61, 53, 45, 37, 29,  21, 13, 5, 28, 20, 12, 4, //2 48  14, 17, 11, 24, 1, 5,  3, 28, 15, 6, 21, 10,  23, 19, 12, 4, 26, 8,  16, 7, 27, 20, 13, 2,  41, 52, 31, 37, 47, 55,  30, 40, 51, 45, 33, 48,  44, 49, 39, 56, 34, 53,  46, 42, 50, 36, 29, 32};  **const int** des\_key\_shift[16] = {  1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};   **const int** des\_expansion[48] = {  32, 1, 2, 3, 4, 5,  4, 5, 6, 7, 8, 9,  8, 9, 10, 11, 12, 13,  12, 13, 14, 15, 16, 17,  16, 17, 18, 19, 20, 21,  20, 21, 22, 23, 24, 25,  24, 25, 26, 27, 28, 29,  28, 29, 30, 31, 32, 1};   **const int** des\_sbox[512] = { //1  14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,  0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,  4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,  15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13, //2  15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,  3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,  0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,  13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9, //3  10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,  13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,  13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,  1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12, //4  7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,  13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,  10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,  3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14, //5  2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,  14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,  4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,  11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3, //6  12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,  10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,  9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,  4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13, //7  4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,  13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,  1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,  6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12, //8  13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,  1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,  7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,  2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};  **const int** des\_pbox[32] = {  16, 7, 20, 21,  29, 12, 28, 17,  1, 15, 23, 26,  5, 18, 31, 10,  2, 8, 24, 14,  32, 27, 3, 9,  19, 13, 30, 6,  22, 11, 4, 25}; #endif //IPR1\_DES\_H |

des.cpp

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/17/17. // #include <cstdint> #include "des.h"  **void** des\_process\_block(uint64\_t\* block, uint64\_t\* keys,**bool** encrypt) {  uint64\_t temp\_block = 0;  //initial permutation  permutation(&temp\_block,\*block,des\_ip,64);   //split  uint32\_t block\_l = 0;  uint32\_t block\_r = 0;  **for**(**int** i = 0; i < 32; i++)  {  set\_bit(&block\_l, temp\_block, i, i);  set\_bit(&block\_r, temp\_block, 32+i, i);  }  **if**(encrypt) {  des\_encrypt\_feistel(&block\_l, &block\_r, 16, keys);  }**else**{  des\_decrypt\_feistel(&block\_l, &block\_r, 16, keys);  }  //join  **for**(**int** i = 0; i < 32; i++)  {  set\_bit(&temp\_block, block\_l, i, i);  set\_bit(&temp\_block, block\_r, i, 32+i);  }  //final permutation  permutation(block,temp\_block,des\_fp,64); }  uint32\_t des\_f\_round(**const** uint32\_t data, uint64\_t key){  //1.  uint64\_t block48 = 0;  des\_expand(&block48,data);  //2.  block48 = block48 ^ key;  //3.  uint32\_t block32 = 0;  substitution(&block32,block48,6,4,8,16,des\_sbox);  //4.  uint32\_t temp\_data = 0;  permutation(&temp\_data,block32,des\_pbox,32);  **return** temp\_data; }  **void** des\_expand(uint64\_t \*target,**const** uint32\_t src){  **for**(**int** i = 0; i < 48; i++){  set\_bit(target,src,des\_expansion[i]-1,i);  } }    **void** des\_encrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint64\_t \* key){  uint32\_t temp = 0;  **for**(**int** i = 0; i<rounds; i++){  temp = \*right ^ des\_f\_round(\*left,key[i+1]);  \*right = \*left;  \*left = temp;  } }  **void** des\_decrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint64\_t \* key){  uint32\_t temp = 0;  **for**(**int** i = rounds; i>0; i--){  temp = \*left ^ des\_f\_round(\*right,key[i]);  \*left = \*right;  \*right = temp;  } }  **void** des\_key\_expansion(uint64\_t\* key, **const** uint64\_t startKey){  **const int** keyblockCount = 7;  **int** keyblockSize = **sizeof**(uint8\_t)\*8;  **for**(**int** i = 0; i<keyblockCount;i++){  **for**(**int** j = 0; j<keyblockSize+1;j++){  **int** to = i\*(keyblockSize+1)+j;  **int** cbit = (i+1)\*keyblockSize-1;  **if**(to<cbit) {  set\_bit(key,startKey,i\*8+j,to);  }**else if**(to == cbit){ // set\_bit(key,1-(startKey[i]%2),0,cbit);  set\_bit(key,1,0,cbit);  }**else**{  set\_bit(key,startKey,i\*8+j-1,to);  }  }  } }  **void** des\_generate\_keys(uint64\_t\* keys,**const** uint64\_t startKey) {  keys[0] = startKey; // des\_key\_expansion(&keys[0],startKey);   uint32\_t\* key\_l = **new** uint32\_t[16];  uint32\_t\* key\_r = **new** uint32\_t[16];   //PC1  uint64\_t pc1key = 0;  permutation(&pc1key,keys[0],des\_key\_permutation,56);  keys[0] = pc1key;  uint32\_t pc1key\_l = 0;  uint32\_t pc1key\_r = 0;  //split into 2 keys by bits  **for**(**int** i = 0; i < 28; i++)  {  set\_bit(&pc1key\_l, keys[0], i, i);  set\_bit(&pc1key\_r, keys[0], 28+i, i);  }   **for** (**int** round = 0; round < 16; round++) {  // 2. key\_shifts  cycle\_shift\_left(&key\_l[round], round == 0 ? pc1key\_l : key\_l[round-1], 28,des\_key\_shift[round]);  cycle\_shift\_left(&key\_r[round], round == 0 ? pc1key\_r : key\_r[round-1], 28,des\_key\_shift[round]);  //3.  //join parts  uint64\_t joinedKey = 0;  **for**(**int** i = 0; i < 28; i++)  {  set\_bit(&joinedKey, key\_l[round], i, i);  set\_bit(&joinedKey, key\_r[round], i, i+28);  }  //4. permute2  **for**(**int** i = 0; i < 48; i++){  set\_bit(&keys[round], joinedKey, des\_key\_permutation[56+i] - 1, i); //used 56 offset to start in 2nd part of perm  } // printf("\n48K%d\n",round); // print\_bits(keys[round],48);  } } |

Gost.h

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/17/17. //  #ifndef IPR1\_GOST\_H #define IPR1\_GOST\_H #include <cstdint> #include "utility.h" **const int** gost28147\_key\_indexes[32] = {  1,2,3,4,5,6,7,8,  1,2,3,4,5,6,7,8,  1,2,3,4,5,6,7,8,  8,7,6,5,4,3,2,1 };  **const int** gost28147\_sbox[128] = {  0x4,0xA,0x9,0x2,0xD,0x8,0x0,0xE,0x6,0xB,0x1,0xC,0x7,0xF,0x5,0x3,  0xE,0xB,0x4,0xC,0x6,0xD,0xF,0xA,0x2,0x3,0x8,0x1,0x0,0x7,0x5,0x9,  0x5,0x8,0x1,0xD,0xA,0x3,0x4,0x2,0xE,0xF,0xC,0x7,0x6,0x0,0x9,0xB,  0x7,0xD,0xA,0x1,0x0,0x8,0x9,0xF,0xE,0x4,0x6,0xC,0xB,0x2,0x5,0x3,  0x6,0xC,0x7,0x1,0x5,0xF,0xD,0x8,0x4,0xA,0x9,0xE,0x0,0x3,0xB,0x2,  0x4,0xB,0xA,0x0,0x7,0x2,0x1,0xD,0x3,0x6,0x8,0x5,0x9,0xC,0xF,0xE,  0xD,0xB,0x4,0x1,0x3,0xF,0x5,0x9,0x0,0xA,0xE,0x7,0x6,0x8,0x2,0xC,  0x1,0xF,0xD,0x0,0x5,0x7,0xA,0x4,0x9,0x2,0x3,0xE,0x6,0xB,0x8,0xC };  **void** gost28147\_process\_block(uint64\_t \*block, uint32\_t \*keys,**bool** encrypt); **void** gost28147\_encrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint32\_t \* key); **void** gost28147\_decrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint32\_t \* key); #endif //IPR1\_GOST\_H |

Gost.cpp

|  |
| --- |
| // // Created by Alexander Degtyarev on 5/17/17. // #include <cstdlib> #include <ctime> #include "gost.h"  uint32\_t\* gost28147\_generateKeys(**char**\* keytext,**int** count) {  uint32\_t\* keys = **new** uint32\_t[count];  **for**(**int** i = 0; i<count; i++){  keys[i] = 0;  **for**(**int** j = 0; j<(**sizeof**(uint32\_t)/**sizeof**(**char**)); j++){  **for**(**int** k = 0; k<**sizeof**(**char**)\*8;k++){  set\_bit(&keys[i],keytext[i\***sizeof**(uint32\_t)+j],k,j\***sizeof**(**char**)\*8+k);  }  }  }  **return** keys; }  uint32\_t\* generateRandomKeys(**int** count){  **auto** keys = **new** uint32\_t[count];  srand((**unsigned int**) time(0));  **for**(**int** i = 0; i<count; i++){  keys[i] = (uint32\_t) rand();  }  **return** keys; }  **void** gost28147\_process\_block(uint64\_t \*block, uint32\_t \*keys,**bool** encrypt) {  //split  uint32\_t block\_l = 0;  uint32\_t block\_r = 0;  **for**(**int** i = 0; i < 32; i++)  {  set\_bit(&block\_l, \*block, i, i);  set\_bit(&block\_r, \*block, 32+i, i);  }  //f  **if**(encrypt) {  gost28147\_encrypt\_feistel(&block\_l, &block\_r, 32, keys);  }**else**{  gost28147\_decrypt\_feistel(&block\_l, &block\_r, 32, keys);  }  //join  **for**(**int** i = 0; i < 32; i++)  {  set\_bit(block, block\_l, i, i);  set\_bit(block, block\_r, i, 32+i);  } }  uint32\_t gost28147\_round(**const** uint32\_t data, uint32\_t key){ // uint64\_t data64 = data; // data64 = data64 + key; // data64 = data64 & 0xFFFFFFFF; // uint32\_t tempBlock = (uint32\_t) data64;  uint32\_t tempBlock = data;  uint32\_t substituted = 0;  substitution(&substituted,tempBlock,4,4,8,16,gost28147\_sbox,**true**);  cycle\_shift\_left(&tempBlock,substituted,32,11);  **return** tempBlock; }  **void** gost28147\_encrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint32\_t \* keys){  uint32\_t temp = 0;  **for**(**int** i = 0; i<rounds; i++){  temp = \*right ^ gost28147\_round(\*left,keys[gost28147\_key\_indexes[i]-1]);  \*right = \*left;  \*left = temp;  } }  **void** gost28147\_decrypt\_feistel(uint32\_t \* left, uint32\_t \* right, **int** rounds, uint32\_t \* keys){  uint32\_t temp = 0;  **for**(**int** i = rounds; i>0; i--){  temp = \*left ^ gost28147\_round(\*right,keys[gost28147\_key\_indexes[i]-1]);  \*left = \*right;  \*right = temp;  } } |

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; } |