МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

ОТЧЁТ

По лабораторной работе № \_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил:

Студент группы ИИ-22

Копанчук Евгений Романович

Проверил\_\_:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Брест – 2023

**Ход работы**

**CBC**

|  |
| --- |
| CBC::CBC() {  block\_size\_ = 0;  v\_init\_ = "";  encode\_matrix\_ = { 2, 1, 3, 2 };  decode\_matrix\_ = { 2, -1, -3, 2 };  }  CBC::CBC(int block\_size, std::string v\_init) : CBC() {  SET\_BLOCK\_SIZE(block\_size);  SET\_VECTOR(v\_init);  }  void CBC::INFO() {  std::cout << "BLOCK SIZE: " << block\_size\_ << std::endl;  std::cout << "INIT VECTOR: " << v\_init\_ << std::endl;  std::cout << "ENCODE MATRIX:";  for (int i = 0; i < 4; i++) {  if (i % 2 == 0)  std::cout << std::endl;  std::cout << encode\_matrix\_[i] << ' ';  }  std::cout << "\nDECODE MATRIX:";  for (int i = 0; i < 4; i++) {  if (i % 2 == 0)  std::cout << std::endl;  std::cout << decode\_matrix\_[i] << ' ';  }  std::cout << std::endl;  }  std::string CBC::ENCODE(std::string message) {  std::string encoded\_message = "";  std::vector<std::string> blocks = SPLIT\_TO\_BLOCKS(message, block\_size\_);  int n = blocks.size();  std::string encoded\_prev = v\_init\_;  for (int i = 0; i < n; i++) {  std::string xor\_res = XOR\_BLOCKS(encoded\_prev, blocks[i]);  std::string encode\_res = ENCODE\_WITH\_MATRIX(xor\_res);  encoded\_prev = encode\_res;  encoded\_message += encode\_res;  }  return encoded\_message;  }  std::string CBC::DECODE(std::string message) {  std::string decoded\_message = "";  std::vector<std::string> blocks = SPLIT\_TO\_BLOCKS(message, block\_size\_);  int n = blocks.size();  std::string encoded\_prev = v\_init\_;  for (int i = 0; i < n; i++) {  std::string decode\_res = DECODE\_WITH\_MATRIX(blocks[i]);  std::string xor\_res = XOR\_BLOCKS(encoded\_prev, decode\_res);  encoded\_prev = blocks[i];  decoded\_message += xor\_res;  }  return decoded\_message;  }  std::string CBC::ENCODE\_WITH\_MATRIX(std::string block) {  std::vector<int> v\_block = STRING\_TO\_INT\_VECTOR(block);  std::vector<int> m1; std::copy(v\_block.begin(), v\_block.begin() + 4, std::back\_inserter(m1));  std::vector<int> m2; std::copy(v\_block.begin() + 4, v\_block.end() - 1, std::back\_inserter(m2));  m1 = SQUARE\_MATRIX\_MULT(encode\_matrix\_, m1);  m2 = SQUARE\_MATRIX\_MULT(encode\_matrix\_, m2);  std::vector<int> encoded\_v\_block = m1;  std::copy(m2.begin(), m2.end(), std::back\_inserter(encoded\_v\_block));  encoded\_v\_block.push\_back(v\_block[block\_size\_ - 1]);  return INT\_VECTOR\_TO\_STRING(encoded\_v\_block);  }  std::string CBC::DECODE\_WITH\_MATRIX(std::string block) {  std::vector<int> v\_block = STRING\_TO\_INT\_VECTOR(block);  std::vector<int> m1; std::copy(v\_block.begin(), v\_block.begin() + 4, std::back\_inserter(m1));  std::vector<int> m2; std::copy(v\_block.begin() + 4, v\_block.end() - 1, std::back\_inserter(m2));  m1 = SQUARE\_MATRIX\_MULT(decode\_matrix\_, m1);  m2 = SQUARE\_MATRIX\_MULT(decode\_matrix\_, m2);  std::vector<int> decoded\_v\_block = m1;  std::copy(m2.begin(), m2.end(), std::back\_inserter(decoded\_v\_block));  decoded\_v\_block.push\_back(v\_block[block\_size\_ - 1]);  return INT\_VECTOR\_TO\_STRING(decoded\_v\_block);  }  std::vector<std::string> CBC::SPLIT\_TO\_BLOCKS(std::string message, int block\_size) {  while (message.size() % block\_size != 0) message += (char)0;  std::vector<std::string> blocks(message.size() / block\_size);  for (int i = 0; i < blocks.size(); i++)  blocks[i] = message.substr(i \* block\_size, block\_size);  return blocks;  }  std::string CBC::XOR\_BLOCKS(std::string v, std::string block) {  std::string temp = "";  for (int i = 0; i < block.size(); i++)  temp += (char)((int)v[i] ^ (int)block[i]);  return temp;  }  std::vector<int> CBC::STRING\_TO\_INT\_VECTOR(std::string str) {  int size = str.size();  std::vector<int> res(size);  for (int i = 0; i < size; i++)  res[i] = (int)str[i];  return res;  }  std::string CBC::INT\_VECTOR\_TO\_STRING(std::vector<int> vec) {  int size = vec.size();  std::string res = "";  for (int i = 0; i < size; i++)  res += (char)vec[i];  return res;  }  std::vector<int> CBC::SQUARE\_MATRIX\_MULT(std::vector<int> m1, std::vector<int> m2) {  std::vector<int> res(4);  for(int i = 0; i < 2; i++)  for (int j = 0; j < 2; j++)  for (int k = 0; k < 2; k++)  res[i \* 2 + j] += m1[i \* 2 + k] \* m2[k \* 2 + j];  return res;  } |

**OUTPUT**

|  |  |
| --- | --- |
|  |  |