Национальный исследовательский университет «МЭИ»

Институт информационных и вычислительных технологий

Кафедра вычислительных технологий

Лабораторная работа №3

по курсу «Защита информации»

Тема: «Изучение алгоритмов хеширования и ЭЦП с использованием криптобиблиотек»

Выполнил

Студент	Швец Григорий Владиславович
Группа	А-06-19
Дата	29.04.2023

Принял

Преподаватель	к.т.н. Андреева Ирина Николаевна
Оценка	
Дата	
Подпись	

Москва 2023

Содержание

Цель работы

Целью данной лабораторной работы является изучение алгоритмов хеширования и ЭЦП с использованием криптобиблиотек.

Задание

1. Изучить материал по алгоритмам хеширования и формирования ЭЦП.
2. Подготовить текстовый файл, содержимое которого будет хешироваться и подписываться ЭЦП в соответствии с пунктами задания.
3. Используя алгоритмы хеширования или ЭЦП (см.табл.), сформировать хеш **или** ЭЦП и записать их в конец файла, **имя которого вводится с клавиатуры**.
4. Внести изменения в исходный текстовый файл и обеспечить возможность поверки нового хеша **или** ЭЦП с записанными в файле, **имя которого вводится с клавиатуры. Предусмотреть** вывод на экран сообщения об отсутствии или наличии изменений в файле, и вывести его актуальное на момент проверки содержимое на экран.

№	Библиотека	ХЕШ-алгоритм	ЭЦП-алгоритм
17	Libgcrypt	Whirlpool	ГОСТ 34.10-2001

Исходный код программ

```
#include <iostream>
#include <gcrypt.h>
#include <fstream>
#include <cstring>
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;

int main() {
    gcry_error_t gcryError;
    gcry_md_hd_t gcryMdHd;
    string InputFile, OutputFile;
    string PrevHashValue;
    bool first_iteration = true;

    while (true) {
        cout << "Please, enter the name of input file or write 'q' to exit: ";
        cin >> InputFile;
        cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~|\n";

        if (InputFile == "q") {
            break;
```

```
        }

        ifstream file(InputFile);

        if (!file) {
            cout << "File open error!\n";
            continue;
        }
        else {
            cout << "All OK! File is open!\n";
            cout <<
"=============================================================================
===|\n";
        }

        ostringstream oss;
        oss << file.rdbuf();
        string str = oss.str();
        const char* plaintext = str.c_str();
        cout << "Input file content: " << str.c_str();
        cout <<
"\n=============================================================================
=====|\n";

        gcryError = gcry_md_open(&gcryMdHd, GCRY_MD_WHIRLPOOL, 0);
        if (gcryError) {
            cout << "gcry_md_open error: " << gcry_strsource(gcryError) << endl;
            return 1;
        }

        gcry_md_write(gcryMdHd, plaintext, strlen(plaintext));

        size_t digest_size = gcry_md_get_algo_dlen(GCRY_MD_WHIRLPOOL);
        cout << "Current digest size is: " << digest_size;
        cout <<
"\n=============================================================================
=====|";
        unsigned char* digest = new unsigned char[digest_size];
        gcry_md_final(gcryMdHd);
        memcpy(digest, gcry_md_read(gcryMdHd, GCRY_MD_WHIRLPOOL), digest_size);

        gcry_md_close(gcryMdHd);

        cout << "\nWhirlpool hash of the plaintext is: ";
        ostringstream output_stream;
        for (int i = 0; i < digest_size; i++) {
            printf("%02x", digest[i]);
            output_stream << hex << setw(2) << setfill('0') <<
static_cast<int>(digest[i]); // Store hex string
        }
        cout << endl;
        cout <<
"=============================================================================
===|\n" << endl;

        if (!first_iteration && PrevHashValue != output_stream.str()) {
            cout << "---ATTENTION!---\nFile content is changed.\n";
        }
        else {
            cout << "File content is not changed.\n";
        }

        cout << "Enter the output file name: ";
        cin >> OutputFile;
        ofstream output(OutputFile);
```

```
        output << output_stream.str();

        delete[] digest;

        PrevHashValue = output_stream.str();
        first_iteration = false;
    }
    return 0;
}




        //// Генерация ключевой пары
        //gcry_sexp_t keypair, public_key, private_key;
        //gcry_error_t error = gcry_sexp_build(&keypair, NULL, "%S");
        //if (error) {
        //    cerr << "Ошибка при генерации ключевой пары: " <<
gcry_strerror(error) << endl;
        //    return 1;
        //}
        //public_key = gcry_sexp_find_token(keypair, "public-key", 0);
        //private_key = gcry_sexp_find_token(keypair, "private-key", 0);

        //// Хэширование данных
        //gcry_md_hd_t hash;
        //const char* data = "gadhgaepojgnqerlkugnqeripgnqpero!";
        //size_t data_length = strlen(data);
        //error = gcry_md_open(&hash, GCRY_MD_GOSTR3411_94, GCRY_MD_FLAG_SECURE);
        //if (error) {
        //    cerr << "Ошибка при открытии контекста хэширования: " <<
gcry_strerror(error) << endl;
        //    return 1;
        //}
        //gcry_md_write(hash, data, data_length);
        //unsigned char* digest = gcry_md_read(hash, GCRY_MD_GOSTR3411_94);
        //size_t digest_length = gcry_md_get_algo_dlen(GCRY_MD_GOSTR3411_94);

        //// Подписание данных
        //gcry_sexp_t signature;
        ////error = gcry_pk_sign(&signature, hash, private_key);
        //gcry_sexp_t hash_sexp;
        //gcry_sexp_build(&hash_sexp, NULL, "(data (flags pkcs1) (hash
GOSTR3411_94 %b))", digest, digest_length);
        //error = gcry_pk_sign(&signature, hash_sexp, private_key);
        ////error = gcry_pk_sign(&signature, digest, digest_length, private_key,
NULL, NULL);
        ////error = gcry_pk_sign(&signature, gcry_md_read(hash,
GCRY_MD_GOSTR3411_94), private_key);
        //if (error) {
        //    cerr << "Ошибка при подписании данных: " << gcry_strerror(error) <<
endl;
        //    return 1;
        //}
        //gcry_sexp_t r = gcry_sexp_find_token(signature, "r", 0);
        //gcry_sexp_t s = gcry_sexp_find_token(signature, "s", 0);
        //const char* r_value = gcry_sexp_nth_data(r, 1, &digest_length);
        //const char* s_value = gcry_sexp_nth_data(s, 1, &digest_length);
        //cout << "Подпись: " << endl;
        //for (size_t i = 0; i < digest_length; i++) {
        //    printf("%02X", r_value[i]);
        //}
        //for (size_t i = 0; i < digest_length; i++) {
        //    printf("%02X", s_value[i]);
```

```
        //}
        //cout << endl;

        //// Проверка подписи
        //gcry_sexp_t data_hash;
        //error = gcry_sexp_build(&data_hash, NULL, "(data (hash GOSTR3411_94
%b))", digest, digest_length);
        //if (error) {
        //    cerr << "Ошибка при построении хэша данных: " <<
gcry_strerror(error) << endl;
        //    return 1;
        //}
        //error = gcry_pk_verify(signature, data_hash, public_key);
```