Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №4 по курсу «Защита информации»

Тема Программа для создания и проверки электронной цифровой подписи

Студент Волков Г.В.

Группа ИУ7-71Б

Оценка (баллы)

Преподаватели Чиж И. С.

Москва — 2023 г.

Введение

Цифровая подпись — это криптографический механизм, который используется для проверки подлинности и целостности цифровых данных. Мы можем рассматривать его как цифровую версию обычных рукописных подписей, но с более высоким уровнем сложности и безопасности.

Выражаясь простыми словами, мы можем описать цифровую подпись как код прикрепленный к сообщению или документу. После его генерации он выступает в качестве доказательства того, что сообщение не было подделано на протяжении своего пути от отправителя к получателю.

Цель — реализация программы создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритмов хеширования MD5.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— изучить алгоритм работы алгоритма RSA;

— изучить алгоритм хеширования MD5;

— реализовать в виде программы алогритм RSA и алгоритм MD5;

— обеспечить шифрование и расшифровку произвольного файла с использованием разработанной программы;

— предусмотреть работу программы с пустым, однобайтовым файлом и с файлами архива (rar, zip или др.).

1 Аналитическая часть

1.1 Алгоритм RSA

Ассиметричный алгоритм криптографии RSA, датой возникновения концепции которого считается 1976 год сейчас очень активно используется для обмена данными, верификацией источника программного обеспечения и в других сферах, где необходимо обмениваться данными или верифицировать отправителя.

В отличии от симметричных алгоритмов шифрования, имеющих всего один ключ для шифрования и расшифровки информации, в алгоритме RSA используется 2 ключа — открытый (публичный) и закрытый (приватный).

Публичный ключ шифрования передаётся по открытым каналам связи, а приватный всегда держится в секрете. В ассиметричной криптографии и алгоритме RSA, в частности, публичный и приватный ключи являются двумя частями одного целого и неразрывны друг с другом.

В основе работы данного алгоритма лежит математический объект, называемый перестановкой с потайным входом — функция, которая преобразует число x в число y в том же диапазоне, так что вычислить y по x легко, зная открытый ключ, но вычислить x по y практически невозможно, если не знать закрытого ключа – потайного входа.(Можете считать, что x – открытый текст, а y – шифртекст.)

Кроме шифрования, RSA используется для создания цифровых подписей, когда только владелец закрытого ключа может подписать сообщение, а наличие открытого ключа позволяет любому желающему проверить достоверность подписи.

Из–за использования возведения в степень данный алгоритм кодирует ноль нулём, поэтому его в основном используют только для передачи ключа симметричного алгоритма. Также недостатком является скорость работы, из–за большого количества операций возведения в степень больших чисел. Стойкость алгоритма базируется на сложности решения задачи факторизации.

На рисунке 1.1 представлен пример работы алгоритма RSA.

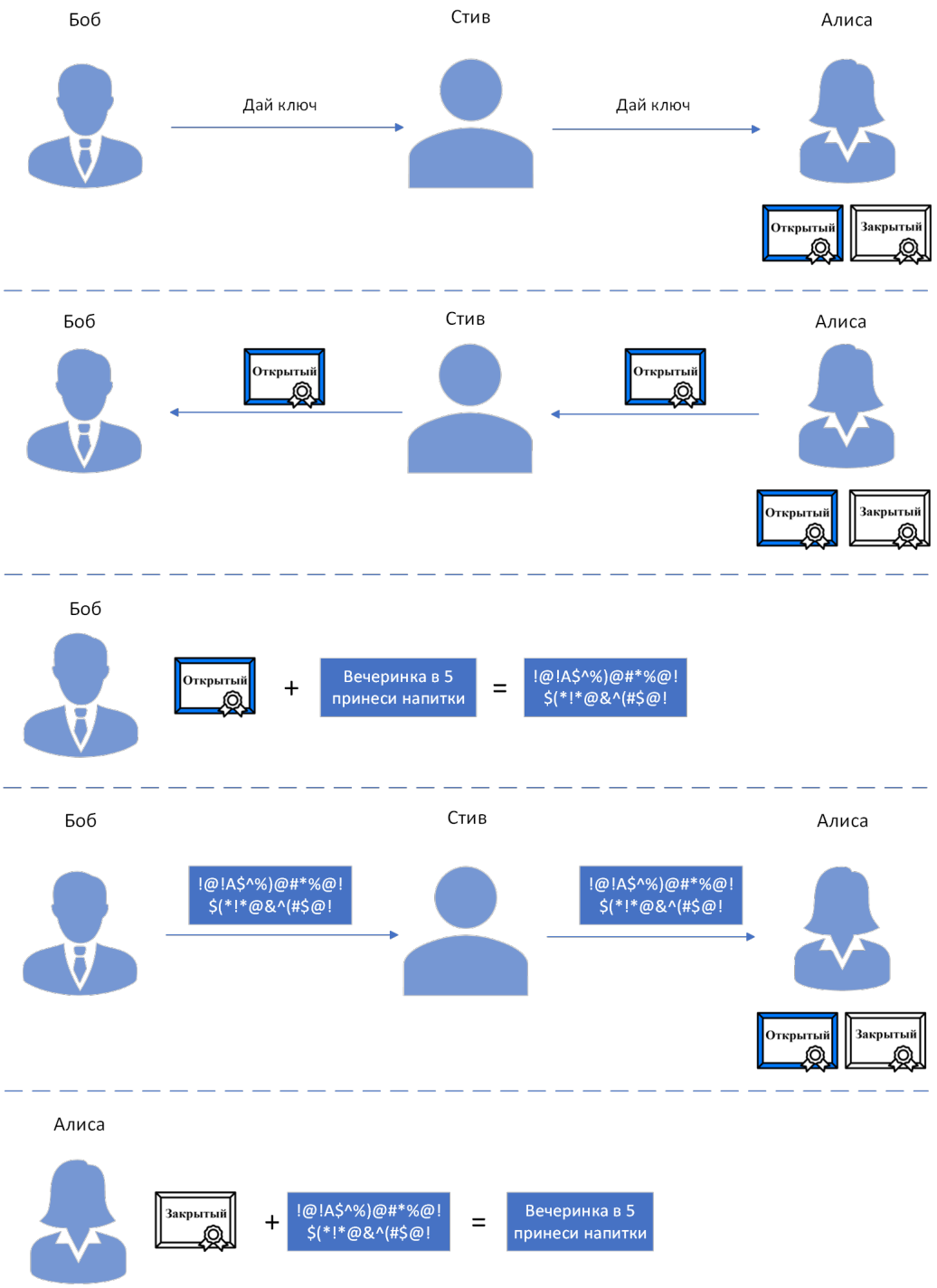

Рисунок 1.1 – Пример работы RSA

Алгоритм RSA видит сообщение как большое число, а само шифрование заключается, по существу, в умножении больших чисел. RSA видит открытый текст как положительное целое число от 1 до $n - 1$, где n – модуль. При перемножении таких чисел получается третье число, удовлетворяющее тем же условиям. Эти числа образуют мультипликативную группу целых чисел по модулю n. Для нахождения числа элементов группы, когда n не является простым числом, используется функция Эйлера $\varphi(n)$. Она даёт количество

чисел, меньших n и взаимно простых с n, т. е. как раз количество элементов группы. Если n разложить в произведение простых чисел $n = p_1 * ... * p_m$, то $\varphi(n) = (p_1-1)*...*(p_m-1)$. RSA имеет дело только с числами n, являющимися произведением двух больших простых чисел, $n = p*q$, следовательно $\varphi(n) = (p-1)*(q-1)$.

Если задан модуль n и число e, называемое открытым показателем степени, то перестановка с потайным входом преобразует число x в $y = x^e mod n$. n и e составляют открытый ключ. Чтобы получить x по y, нам нужно еще одно число, d, такое что: $x = y^d mod n = (x^e mod n)^d mod n = x^{ed} mod n = x$. Закрытый ключ состоит из n и d.

Очевидно, что d – не любое число а такое, что $e * d = 1$. Точнее, должно иметь место равенство $ed = 1 mod \varphi(n)$. Заметим, что вычисление производится по модулю $\varphi(n)$, а не по модулю n, потому что показатели степени ведут себя как индексы элементов, которых всего $\varphi(n)$.

1.1.1 Генерация ключей

Процедура создания публичного и приватного ключей:

— выбираем два случайных простых числа p и q;

— вычисляем их произведение: $n = p * q$;

— вычисляем функцию Эйлера: $\varphi(n) = (p-1)*(q-1)$;

— выбираем число e, которое меньше $\varphi(n)$ и является взаимно простым с $\varphi(n)$;

— ищем число d, обратное числу e по модулю $\varphi(n)$, т.е. остаток от деления $(d*e)$ и $\varphi(n)$ должен быть равен 1. Найти его можно через расширенный алгоритм Евклида.

После произведённых вычислений получаем публичный ключ $\{e, n\}$ и приватный ключ $\{d, n\}$. Желательно брать e так, чтоб в его двоичной записи было минимальное количество единиц, что позволить совершать меньше операций в алгоритме быстрого возведения в степень.

1.2 Алгоритм хеширования MD5

Хэш-функция предназначена для свертки входного массива любого размера в битовую строку, для MD5 длина выходной строки равна 128 битам. Если имеется два массива, и необходимо быстро сравнить их на равенство, то хэш-функция поможет сделать это, если у двух массивов хэши разные, то массивы гарантировано разные, а в случае равенства хэшей — массивы скорее всего равны. Однако чаще всего хэш-функции используются для проверки уникальности пароля, файла, строки и тд.

Алгоритм включает в себя 5 основных действий:

— выравнивание потока. Сначала к концу потока дописывают единичный бит. Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока стала сравнима с 448 по модулю 512. Необходимо для следующего этапа;

— добавление длины сообщения. В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. осле этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит;

— инициализация буфера. Для вычислений инициализируются четыре переменные размером по 32 бита, в этих переменных будут храниться результаты промежуточных вычислений.

— вычисление в цикле. Происходит перемешивание 16 раундов по 4 этапа;

— результат вычислений.

На рисунке 1.2 представлен пример работы алгоритма хеширования MD5.

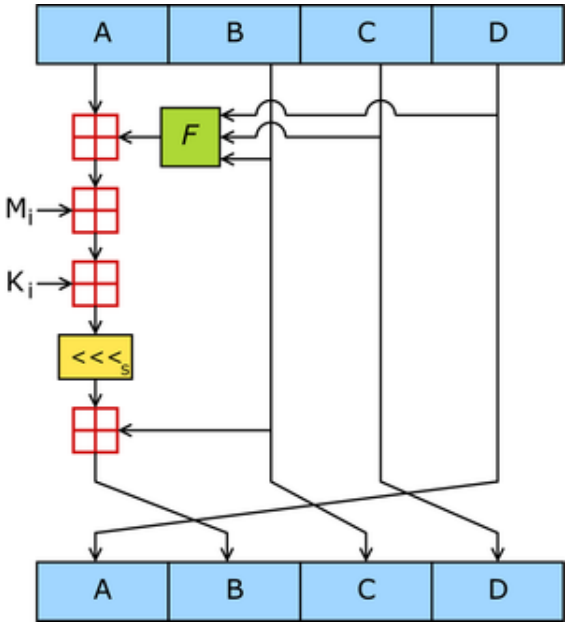

Рисунок 1.2 – Пример работы MD5

1.3 Электронно цифровая подпись

Система RSA может использоваться не только для шифрования, но и для цифровой подписи. Предположим, что Алисе нужно отправить Бобу сообщение m, подтверждённое электронной цифровой подписью. Схема обема сообщениями с ЭЦП представлена на рисунке 1.3.

Рисунок 1.3 – Схема работы ЭЦП

Действия Алисы:

— взять открытый текст m;

— создать ЭЦП с помощью своего секретного ключа, получить хэш файла и зашифровать его;

— передать пару из файла и подписи.

Действия Боба:

— принять пару из файла и подписи;

— расшифровать подпись открытым колючем;

— получить хэш принятого файла;

— проверить равенство двух хешей.

Важное свойство цифровой подписи заключается в том, что её может проверить каждый, кто имеет доступ к открытому ключу её автора. Один из участников обмена сообщениями после проверки подлинности цифровой подписи может передать подписанное сообщение ещё кому-то, кто тоже в состоянии проверить эту подпись.

2 Конструкторская часть

В этом разделе будут представлены сведения о модулях программы.

2.1 Сведения о модулях программы

Программа состоит из двух модулей:

1) $main.c$ — файл, содержащий точку входа;

2) $md5.c$ — файл, содержащий алгоритм хеширования MD5;

3) $md5.h$ — файл, содержащий описание функций для алгоритма хеширования MD5;

4) $rsa.c$ — файл, содержащий функции генерирования ключей и алгоритма RSA;

5) $rsa.h$ — файл, содержащий описания функций генерирования ключей и алгоритма RSA.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, представлены листинги реализаций алгоритма шифрования RSA и результаты тестирования.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования C. Данный язык удовлетворяет поставленным критериям по средствам реализации.

3.2 Реализация алгоритма

В листинге 3.1 представлена реализация алгоритма шифрования RSA.

Листинг 3.1 – RSA

```
1  lli gcde(lli a, lli b, lli *x, lli *y) {
2      if (a == 0) {
3          *x = 0;
4          *y = 1;
5          return b;
6      }
7      lli x1, y1;
8      lli d = gcde(b % a, a, &x1, &y1);
9      *x = y1 - (b / a) * x1;
10     *y = x1;
11     return d;
12 }
13
14 lli mod_exp(lli num, lli exp, lli mod) {
15     lli product = 1;
16
17     while (exp > 0) {
18         if (exp & 0x01) product = (product * num) % mod;
19         num = (num * num) % mod;
```

```
20          exp >>= 1;
21      }
22
23      return product;
24  }
25
26  void rsa_key_gen(rsa_key_t *public_key, rsa_key_t *private_key) {
27      lli N = 0, phi = 0;
28      lli e = E;
29
30      N = P * Q;
31      phi = (P - 1) * (Q - 1);
32
33      lli x, y;
34      gcde(e, phi, &x, &y);
35      while (x < 0) x = x + phi;
36
37      public_key->mod = N;
38      public_key->exp = e;
39
40      private_key->mod = N;
41      private_key->exp = x;
42  }
43
44  lli rsa(const lli data, const struct rsa_key_t key) {
45      return mod_exp(data, key.exp, key.mod);
46  }
```

3.3 Тестирование

Для тестирование написанной программы сначала генерировались ключи. Потом подписывался файл и проверялась его подпись.

Таблица 3.1 – Функциональные тесты ЭЦП

Входной файл	Ожидаемый результат	Результат
Текстовый файл	Успешная проверка	Успешная проверка
1 байтовый файл	Успешная проверка	Успешная проверка
Пустой файл	Успешная проверка	Успешная проверка
Архив	Успешная проверка	Успешная проверка
PNG изображение	Успешная проверка	Успешная проверка
2 разных файла	Проваленная проверка	Проваленная проверка

Для тестирование MD5 был введён дополнительный режим работы программы, позволяющий получить хеш файла. Результат работы программы сравнивался с результатом md5sum с помощью команды cmp.

Таблица 3.2 – Функциональные тесты MD5

Входной файл	Ожидаемый результат	Результат
Текстовый файл	Пустой вывод	Пустой вывод
1 байтовый файл	Пустой вывод	Пустой вывод
Пустой файл	Пустой вывод	Пустой вывод
Архив	Пустой вывод	Пустой вывод
PNG изображение	Пустой вывод	Пустой вывод

Заключение

В результате лабораторной работы были изучены принципы работы алгоритма RSA и алгоритма хеширования MD5, была реализована программа, способная работать с электронной цифровой подписью.

Были решены следующие задачи:

1) изучен алгоритм работы алгоритма RSA;

2) изучен алгоритм хеширования MD5;

3) реализован в виде программы алогритм RSA и алгоритм MD5;

4) обеспечино шифрование и расшифровку произвольного файла с использованием разработанной программы;

5) предусмотрена работа программы с пустым, однобайтовым файлом и с файлами архива (rar, zip или др.).