Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 6

“Компьютерная реализация электронной цифровой подписи”

Выполнил студент гр. 753504

Горбачёнок К. Н.

Проверил

Протько М.И.

Минск, 2020

# Постановка задачи

Реализовать программные средства формирования электронной цифровой подписи на примере алгоритма ГОСТ 3410.

# Описание алгоритма

Электронно-цифровая подпись (ЭЦП) - это реквизит электронного документа, предназначенный для защиты данного электронного документа от подделки, полученный в результате криптографического преобразования информации с использованием закрытого ключа электронной цифровой подписи и позволяющий идентифицировать владельца сертификата ключа подписи, а также установить отсутствие искажения информации в электронном документе.

Электронно-цифровая подпись - это программно-криптографическое средство, которое обеспечивает:

* проверку целостности документов;
* конфиденциальность документов;
* установление лица, отправившего документ.

Преимущества использования электронно-цифровой подписи:

* значительно сократить время, затрачиваемое на оформление сделки и обмен документацией;
* усовершенствовать и удешевить процедуру подготовки, доставки, учета и хранения документов;
* гарантировать достоверность документации;
* минимизировать риск финансовых потерь за счет повышения конфиденциальности информационного обмена;
* построить корпоративную систему обмена документами.

В рамках лабораторной работы необходимо реализовать программные средства проверки ЭЦП на базе алгоритма ГОСТ 3410.

В стандарте *ГОСТ 3410* используется хэш-функция ГОСТ 3411, которая создает хэш-код длиной 256 бит. Это во многом обуславливает требования к выбираемым простым числам p и q:

р должно быть простым числом в диапазоне

2509 < p < 2512либо 21020 < p < 21024

q должно быть простым числом в диапазоне 2254 < q < 2256

q также должно быть делителем (р-1).

Аналогично выбирается и параметр g.

При этом требуется, чтобы gq (mod p) = 1.

В соответствии с теоремой Ферма это эквивалентно условию в *DSS*,

что g = h(p-1)/q mod p.

Закрытым ключом является произвольное число х: 0 < x < q

Открытым ключом является число y: y = gx mod p

Для создания подписи выбирается случайное число k: 0 < k < q

Подпись состоит из двух чисел (r, s), вычисляемых по следующим формулам:

r = (gk mod p) mod q

s = (k H(M) + xr) mod q

Алгоритм *ГОСТ 3411* является отечественным стандартом для хэш-функций. Длина хэш-кода, создаваемого алгоритмом *ГОСТ 3411*, равна 256 битам. Алгоритм разбивает сообщение на блоки, длина которых также равна 256 битам. Кроме того, параметром алгоритма является стартовый вектор хэширования *Н* - произвольное фиксированное значение длиной также 256 бит.

Сообщение обрабатывается блоками по 256 бит справа налево.

Каждый блок сообщения обрабатывается по следующему алгоритму.

1.           Генерация четырех ключей длиной 256 бит каждый.

2.           Шифрование 64-битных значений промежуточного хэш-кода H на ключах Ki(i = 1, 2, 3, 4) с использованием алгоритма ГОСТ 28147 в режиме простой замены.

3.           Перемешивание результата шифрования.

Для генерации ключей используются следующие данные:

·                промежуточное значение хэш-кода Н длиной 256 бит;

·                текущий обрабатываемый блок сообщения М длиной 256 бит;

·                параметры - три значения С2, С3 и С4 длиной 256 бит следующего вида: С2 и С4 состоят из одних нулей, а С3 равно

18 08 116 024 116 08 (08 18)2 18 08 (08 18)4 (18 08)4

где степень обозначает количество повторений 0 или 1.

Используются две формулы, определяющие перестановку и сдвиг.

Перестановка Р битов определяется следующим образом: каждое 256-битное значение рассматривается как последовательность тридцати двух 8-битных значений.

Перестановка Р элементов 256-битной последовательности выполняется по формуле y = φ(x), где x - порядковый номер 8-битного значения в исходной последовательности; y - порядковый номер 8-битного значения в результирующей последовательности.

φ(i + 1 + 4(k - 1)) = 8i + k ; i = 0 ÷ 3, k = 1 ÷ 8

Сдвиг А определяется по формуле

A (x) = (x1  x2) || x4 || x3 || x2

Где xi - соответствующие 64 бита 256-битного значения х, || обозначает конкатенацию.

Присваиваются следующие начальные значения:

i = 1, U = H, V = M.

W = U V, K1 = Р (W)

Ключи K2, K3, K4 вычисляются последовательно по следующему алгоритму:

U = A(U)  Сi, V = A(A(V)),  W = U  V,  Ki = Р(W)

Далее выполняется шифрование 64-битных элементов текущего значения хэш-кода Н с ключами K1, K2, K3 и K4. При этом хэш-код Н рассматривается как последовательность 64-битных значений:

H = h4 || h3 || h2 || h1

Выполняется шифрование алгоритмом ГОСТ 28147:

si = EKi [hi]    i = 1, 2, 3, 4

S = s1 || s2 || s3 || s4

Наконец на заключительном этапе обработки очередного блока выполняется перемешивание полученной последовательности. 256-битное значение рассматривается как последовательность шестнадцати 16-битных значений. Сдвиг обозначается Ψ и определяется следующим образом:

|  |
| --- |
| η16 || η15 || ... || η1 - исходное значение |
| η1η2η3η4η13η16 || η16 || ... || η2 - результирующее значение |

Результирующее значение хэш-кода определяется следующим образом:

Χ(M, H) = ψ61 (H   ψ (M  ψ12(S)))

где

|  |
| --- |
| H - предыдущее значение хэш-кода, |
| М - текущий обрабатываемый блок, |
| Ψi - i-ая степень преобразования Ψ. |

Входными параметрами алгоритма являются:

·                исходное сообщение М произвольной длины;

·                стартовый вектор хэширования Н, длина которого равна 256 битам;

·                контрольная сумма Σ, начальное значение которой равно нулю и длина равна 256 битам;

·                переменная L, начальное значение которой равно длине сообщения.

Сообщение М делится на блоки длиной 256 бит и обрабатывается справа налево. Очередной блок i обрабатывается следующим образом:

1.           H = Χ(Mi, H)

2.           Σ = Σ  ' Mi

3.           L рассматривается как неотрицательное целое число, к этому числу прибавляется 256 и вычисляется остаток от деления получившегося числа на 2256. Результат присваивается L.

Где ' обозначает следующую операцию: Σ и Mi рассматриваются как неотрицательные целые числа длиной 256 бит. Выполняется обычное сложение этих чисел и находится остаток от деления результата сложения на 2256. Этот остаток и является результатом операции.

Самый левый, т.е. самый последний блок М' обрабатывается так:

1.           Блок добавляется слева нулями так, чтобы его длина стала равна 256 битам.

2.           Вычисляется Σ = Σ  ' Mi.

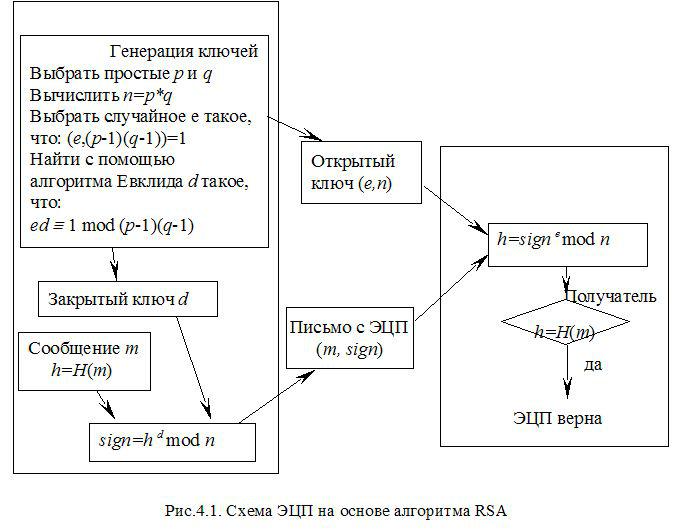
3.           L рассматривается как неотрицательное целое число, к этому числу прибавляется длина исходного сообщения М и находится остаток от деления результата сложения на 2256.

4.           Вычисляется Н = Χ(М', Н).

5.           Вычисляется Н = Χ(L, Н).

6.           Вычисляется Н = Χ(Σ, Н).

# Блок-схема алгоритма

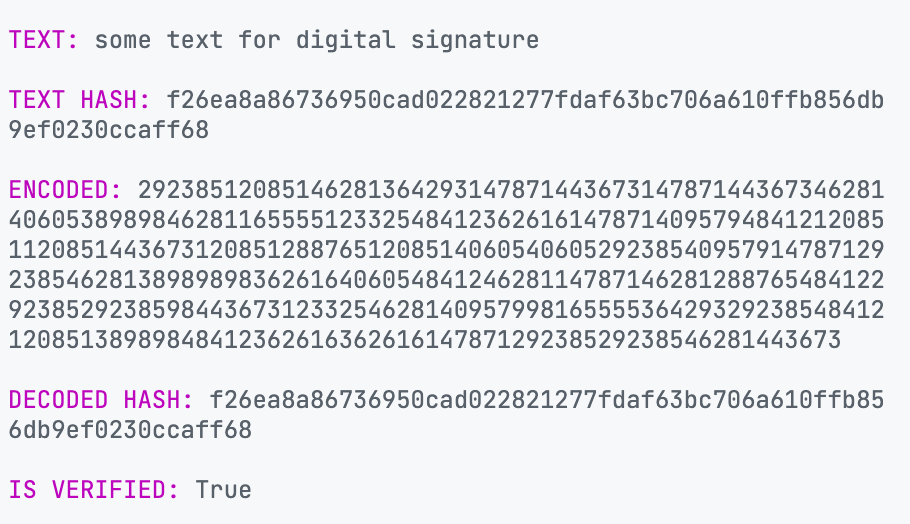


***Блок-схема алгоритма ГОСТ 3410***

# Результаты работы программы

# 

***Исходный файл***

******

***Результат работы программы***

# Прграммный код

import random

from hashlib import sha256

def co\_prime(a, b):

while b != 0:

a, b = b, a % b

return a

def extended\_gcd(aa, bb):

lastremainder, remainder = abs(aa), abs(bb)

x, lastx, y, lasty = 0, 1, 1, 0

while remainder:

lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)

x, lastx = lastx - quotient \* x, x

y, lasty = lasty - quotient \* y, y

return lastremainder, lastx \* (-1 if aa < 0 else 1), lasty \* (-1 if bb < 0 else 1)

def mod\_inv(a, m):

g, x, y = extended\_gcd(a, m)

if g != 1:

raise Exception('Modular inverse does not exist')

return x % m

def is\_prime(num):

if num == 2:

return True

if num < 2 or num % 2 == 0:

return False

for n in range(3, int(num \*\* 0.5) + 2, 2):

if num % n == 0:

return False

return True

def generate\_key\_pair(p, q):

if not (is\_prime(p) and is\_prime(q)):

raise ValueError('Both numbers must be prime.')

elif p == q:

raise ValueError('p and q cannot be equal')

n = p \* q

phi = (p - 1) \* (q - 1)

e = random.randrange(1, phi)

g = co\_prime(e, phi)

while g != 1:

e = random.randrange(1, phi)

g = co\_prime(e, phi)

d = mod\_inv(e, phi)

return (e, n), (d, n)

def encode(private\_key, hashed\_text):

key, n = private\_key

return [pow(ord(char), key, n) for char in hashed\_text]

def decode(public\_key, cipher\_text):

key, n = public\_key

return ''.join([chr(pow(char, key, n)) for char in cipher\_text])

def hash\_function(message):

hashed = sha256(message.encode("UTF-8")).hexdigest()

return hashed

def verify(received\_hashed, message):

our\_hashed = hash\_function(message)

if received\_hashed == our\_hashed:

return True

else:

return False

import random

from algorithms import digital\_signature

from helpers.colored\_text import colored\_text

from helpers.file\_info import read\_text

primes = [i for i in range(0, 1000) if digital\_signature.is\_prime(i)]

def main():

p, q = random.sample(primes, 2)

public\_key, private\_key = digital\_signature.generate\_key\_pair(p, q)

text = read\_text(filename="resources/lab6.txt")

print(colored\_text("TEXT:"), text)

print(colored\_text("TEXT HASH:"), digital\_signature.hash\_function(text))

encoded = digital\_signature.encode(

private\_key=private\_key,

hashed\_text=digital\_signature.hash\_function(text)

)

print(colored\_text("ENCODED:"), ''.join(map(lambda x: str(x), encoded)))

decoded = digital\_signature.decode(

public\_key=public\_key,

cipher\_text=encoded

)

print(colored\_text("DECODED HASH:"), decoded)

print(colored\_text("IS VERIFIED:"), digital\_signature.verify(decoded, text))

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Вывод

В ходе написания лабораторной работы было изучено понятие цифровой подписи с помощью функции хеширования ГОСТ 3410, а также написана программная реализация.