

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет безопасности информационных технологий

Дисциплина: «Стандарты криптографии»

О Т Ч Е Т

ЛАБОРАТОРНАЯ РАБОТА

«Разработка программной реализации ГОСТ 34.10-2012»

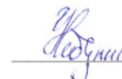
Выполнили студенты группы N3451: Каранкевич В.М.



Мухамеджанов С.



Чебунин К.О.



Яресько С.



Проверил

ассистент ФБИТ,
Университет ИТМО:

Калабишка М.М.

Дата: _____

Санкт-Петербург, 2021

Цель работы

Разработать программную реализацию ГОСТ 34.10–2012 «Процессы формирования и проверки электронной цифровой подписи».

Задачи работы

- провести анализ нормативного документа ГОСТ 34.10–2012;
- на основе проведённого анализа разработать описание алгоритма;
- разработать программную реализацию алгоритма;
- проверить работоспособность алгоритма.

Ход работы

Описанная в стандарте ГОСТ 34.10–2012 модель цифровой подписи охватывает процессы формирования и проверки подписи. Цифровая подпись предназначена для аутентификации лица, подписавшего электронное сообщение. Кроме того, использование ЭЦП предоставляет возможность обеспечить следующие свойства при передаче в системе подписанного сообщения:

- осуществление контроля целостности передаваемого подписанного сообщения;
- доказательное подтверждение авторства лица, подписавшего сообщение;
- защита сообщения от возможной подделки.

Данная криптосистема построена на эллиптической кривой. Эллиптической кривой над конечным простым полем F_p является множество пар чисел (x, y) , для которых выполняется тождество:

$$y^2 = x^3 + ax + b \pmod{p} \quad (1)$$

Для задания эллиптической кривой необходимо выбрать её исходные значения, для этого необходимо задать простое число p – модуль эллиптической кривой такое, что $p > 2^{255}$. Также вычислить её инвариант или соответствующие коэффициенты a, b .

Для вычисления инварианта $J(E)$ используется тождество

$$J(E) = 1728 \frac{4a^3}{4a^3 + 27b^2} \pmod{p} \quad (2)$$

На множестве точек, определённых данной эллиптической кривой, также определена операция сложения (+) и определён нулевой элемент (O).

На основании заданной эллиптической кривой задаётся циклическая подгруппа группы точек заданной эллиптической кривой E с порядком q, где q – простое число, для которого выполняется неравенство $2^{254} < q < 2^{256}$.

Также необходимо вычислить точку P не равную 0, для которой выполняется тождество $qP=0$.

Для формирования и проверки подписи используются ключ подписи – положительное целое число d меньше, чем q, а также ключ проверки подписи – точка эллиптической кривой Q, для которой выполняется тождество $dP=Q$.

Процесс генерации подписи начинается с генерации сообщения, которое может быть хешировано с помощью стандарта ГОСТ 34.11-2012. После этого для блоков размерностью 256 бит вычисляется целое число α , для которого вектор \bar{h} является двоичным представлением и найти число e, для которого:

$$e=\alpha(\bmod q). \quad (3)$$

Далее генерируется число положительное целое число k меньше чем q, на основании которого находится точка $C=kP$. Координата x данной точки позволяет вычислить коэффициент r:

$$r=x_c(\bmod q). \quad (4)$$

Также вычисляется значение параметра s:

$$s=(rd + ke)(\bmod q). \quad (5)$$

На основании двоичных векторов \bar{r} и \bar{s} , полученных на основании коэффициентов r и s соответственно при помощи конкатенации вычисляется значение цифровой подписи – ζ .

Исходными данными данной функции являются ключ подписи и само исходное сообщение. Выходным результатом является сама подпись.

На вход функции проверки цифровой подписи поступает сообщение, цифровая подпись и ключ проверки подписи. На основании цифровой подписи ζ вычисляются значения коэффициентов r и s, причём оба эти коэффициента должны быть неотрицательными и меньшими чем q. В противном случае подпись считается неверной. При использовании хэш-функции из ГОСТ 34.11–2018 необходимо посчитать хэш-код сообщения. Далее необходимо

вычислить целое число a , для которого вектор \bar{h} является двоичным представлением и найти число e , для которого:

$$e \equiv \alpha \pmod{q}. \quad (6)$$

Далее вычисляется значение коэффициента v :

$$v=e^{-1}(\text{mod } q). \quad (7)$$

На основании этих параметров вычисляется значение коэффициентов z_1 и z_2 по формулам:

$$z_1 = sv \pmod{q}; \quad (8)$$

$$z_2 = -rv \pmod{q}. \quad (9)$$

На основании этих коэффициентов можно вычислить точку эллиптической кривой по тождеству:

$$C = z_1 P + z_2 Q. \quad (10)$$

И исходя из x-координаты полученной точки вычисляется значение коэффициента R по формуле:

$$R = x_c(\text{mod } q). \quad (11)$$

Если выполняется равенство $R=r$, то подпись верна, в противном случае, подпись является неверной.

В качестве языка программирования был выбран язык Python в связи с его лёгким доступом к большим типам переменных. Исходный код разработанного алгоритма с комментариями находится в приложении.

Пример работы разработанного алгоритма:

Для корректной работы алгоритма необходимо задать исходные данные эллиптической кривой, а именно параметры p , a , b , координаты точки P , ключ подписи d . Данные параметры были заполнены на основании примеров работы алгоритма из ГОСТ 34.10-2012:

[illegible]

Рисунок 1 – Исходные данные

Также необходимо задать исходное сообщение в бинарном виде:

```
digest=b"Default Message"
```

Рисунок 2 – Исходное сообщение

На основании исходных данных формируется итоговая подпись:

```
C:\Users\konst\Desktop>py "Final 34.10.py"
Исходное сообщение:
b'Default Message'
Итоговая цифровая подпись:
2be7e437b30f6e120598dcf6872509dba606304cb5529f9d1d627cce3e7a210864ffcf297cb19b443a9288522b3cdc7007
cc56123c205a7aba3be22b9e2b64d0
```

Рисунок 3 – Пример работы алгоритма подписи

Далее необходимо проверить работоспособность алгоритма проверки цифровой подписи. При вводе верной цифровой подписи к исходному сообщению программа выдаёт верность подписи. В противном случае, если изменить любой бит подписи подпись становится неверной.

```
C:\Users\konst\Desktop>py "Final 34.10.py"
Исходное сообщение:
b'Default Message'
Исходная цифровая подпись:
03bd85ee302ebfd84389e9f72f9544c07164b39f324e3d989dfeda75be48c45402aec9f59b4c3bb13c371fb4c5009cdc72
d85b4131fd970bdd9e07ba451d350a
Подпись верна
```

Рисунок 4 – Работа алгоритма с верной подписью

```
C:\Users\konst\Desktop>py "Final 34.10.py"
Исходное сообщение:
b'Default Message'
Исходная цифровая подпись:
28bbe74789599eea647af43b5f7c1d90a093505898b2fe5fd34cb05db8d244de248f680859092b619e49691a4ab59fb93e
82bbeb7390ca6fc4b9b87589876e65
Подпись не верна
```

Рисунок 5 - Работа алгоритма с неверной подписью

Выводы

В данной лабораторной работе мы познакомились с алгоритмом цифровой подписи из стандарта ГОСТ 34.10–2012 «Процессы формирования и проверки электронной цифровой подписи», а также проанализировав работу представленного в нормативном документе алгоритма разработали программную реализацию данного алгоритма на языке Python.

[illegible]

```

y=bytes2long(hexdec("08E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B96ABBCEA7
E8FC8"))      #Координата у точки P
prv =
bytes2long(hexdec("7A929ADE789BB9BE10ED359DD39A72C11B60961F49397EEE1D19CE9891EC3
B28"))      #Ключ подписи

def pos(v):
    # Перевод отрицательного числа в положительное
    if v < 0:
        return v + p
    return v

def _add(p1x, p1y, p2x, p2y):
    # Функция сложения точек
    if p1x == p2x and p1y == p2y:
        t = ((3 * p1x * p1x + a) * modinvert(2 * p1y, p)) % p
    else:
        tx = pos(p2x - p1x) % p
        ty = pos(p2y - p1y) % p
        t = (ty * modinvert(tx, p)) % p
    tx = pos(t * t - p1x - p2x) % p
    ty = pos(t * (p1x - tx) - p1y) % p
    return tx, ty

def exp(degree, xx=None, yy=None):
    # Функция вычисления координат кратной точки
    xx = xx or x
    yy = yy or y
    tx = xx
    ty = yy
    if degree == 0:
        raise ValueError("Bad degree value")
    degree -= 1
    while degree != 0:
        if degree & 1 == 1:
            tx, ty = _add(tx, ty, xx, yy)
            degree = degree >> 1
        xx, yy = _add(xx, yy, xx, yy)
    return tx, ty

# Генерация публичного ключа из приватного
pub=exp(prv)
size = point_size(pub[0])
final_pub=(long2bytes(pub[1], size) + long2bytes(pub[0], size))[::-1]

size = point_size(p)
# Информация, которая будет подписываться
digest=b"Default Message"
print("Исходное сообщение:")
print(digest)

# Процедура подписи информации
e = bytes2long(digest) % q #Вычисление числа e, являющееся двоичным
представлением исходного сообщения и числа альфа
if e == 0:
    e = 1
rand=None

while True:
    rand = urandom(size)
    k = bytes2long(rand) % q #Генерация случайного числа k
    if k == 0:

```

```

        continue

    r, _ = exp(k) #Вычисление координаты x точки эллиптической кривой  $C=kP$ 
    r %= q
    if r == 0:
        continue

    s = (prv*r + k*e) % q #Вычисление значения s
    if s == 0:
        continue
    break
signed_data = long2bytes(s, size) + long2bytes(r, size) #Получение подписи через
конкатенацию значений r и s

print("Итоговая цифровая подпись:")
print((hexenc(signed_data)))

# Процедура проверки подписи
size = point_size(p)
if len(signed_data) != size * 2:
    raise ValueError("Invalid signed_data length")

s = bytes2long(signed_data[:size]) #Вычисление значений s и r по значению
подписи и проверка их верности
r = bytes2long(signed_data[size:])
if r <= 0 or r >= q or s <= 0 or s >= q:
    print("Подпись не верна")

e = bytes2long(digest) % q #Вычисление числа e, являющееся двоичным
представлением исходного сообщения и числа альфа
if e == 0:
    e = 1

v = modinvert(e, q) #Вычисление числа v

z1 = s * v % q #Вычисление чисел z1 и z2
z2 = q - r * v % q

p1x, p1y = exp(z1) #Вычисление координаты точки  $C1=z1P$ 
q1x, q1y = exp(z2, pub[0], pub[1]) #Вычисление координаты точки  $C2=z2Q$ 

# Вычисление координаты  $C=C1+C2$ 
lm = q1x - p1x
if lm < 0:
    lm += p
lm = modinvert(lm, p)
z1 = q1y - p1y
lm = lm * z1 % p
lm = lm * lm % p
lm = lm - p1x - q1x
lm = lm % p
if lm < 0:
    lm += p
lm %= q

# Проверка подписи
if lm==r:
    print("Подпись верна")
else:
    print("Подпись не верна")

```