БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра информатики

Факультет: ИНО
Специальность: ИиТП

Контрольная работа № 2

по дисциплине

“Методы защиты информации”

“Алгоритм цифровой подписи ГОСТ 3410”

Выполнил студент: Дубейковский А.А.
Группа № 893551
Зачётная книжка № 75350046

Условие

Контрольная работа №2

Указания по выбору варианта

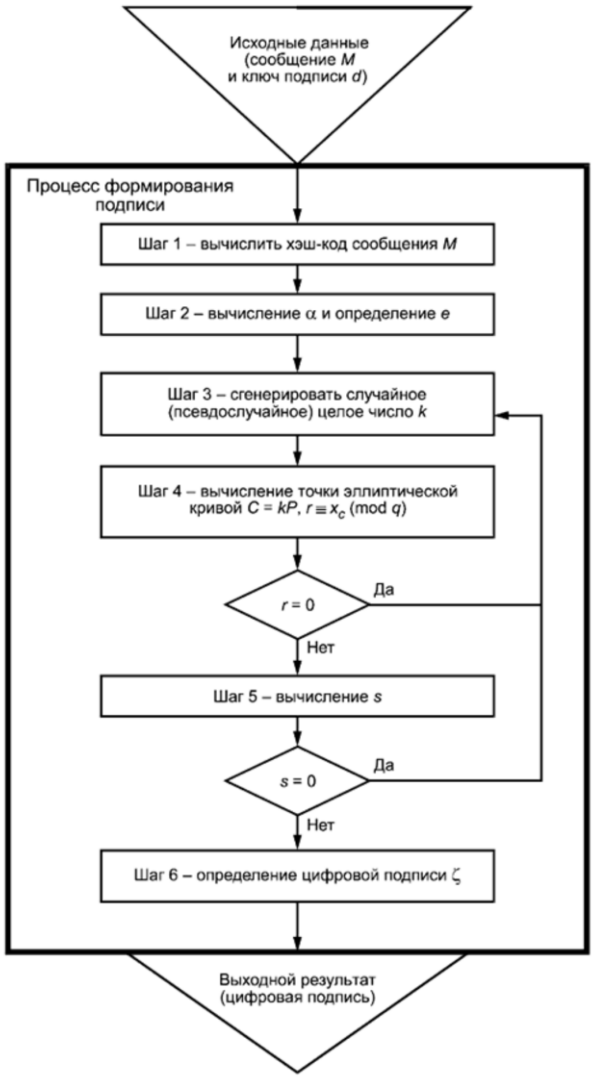
Рабочей программой дисциплины «Методы защиты информации» предусмотрено выполнение двух контрольных работ. Контрольная работа № 2 подразумевает изучение и программную реализацию (на языке высокого уровня) алгоритма стандарта цифровой подписи ГОСТ 3410. В качестве отчета по контрольной работе высылается листинг программной реализации. представлена в виде теста и исполняемый файл. В контрольной работе № 2 используется **один вариант** (для всех номеров зачетных книжек).

Теоретическая часть

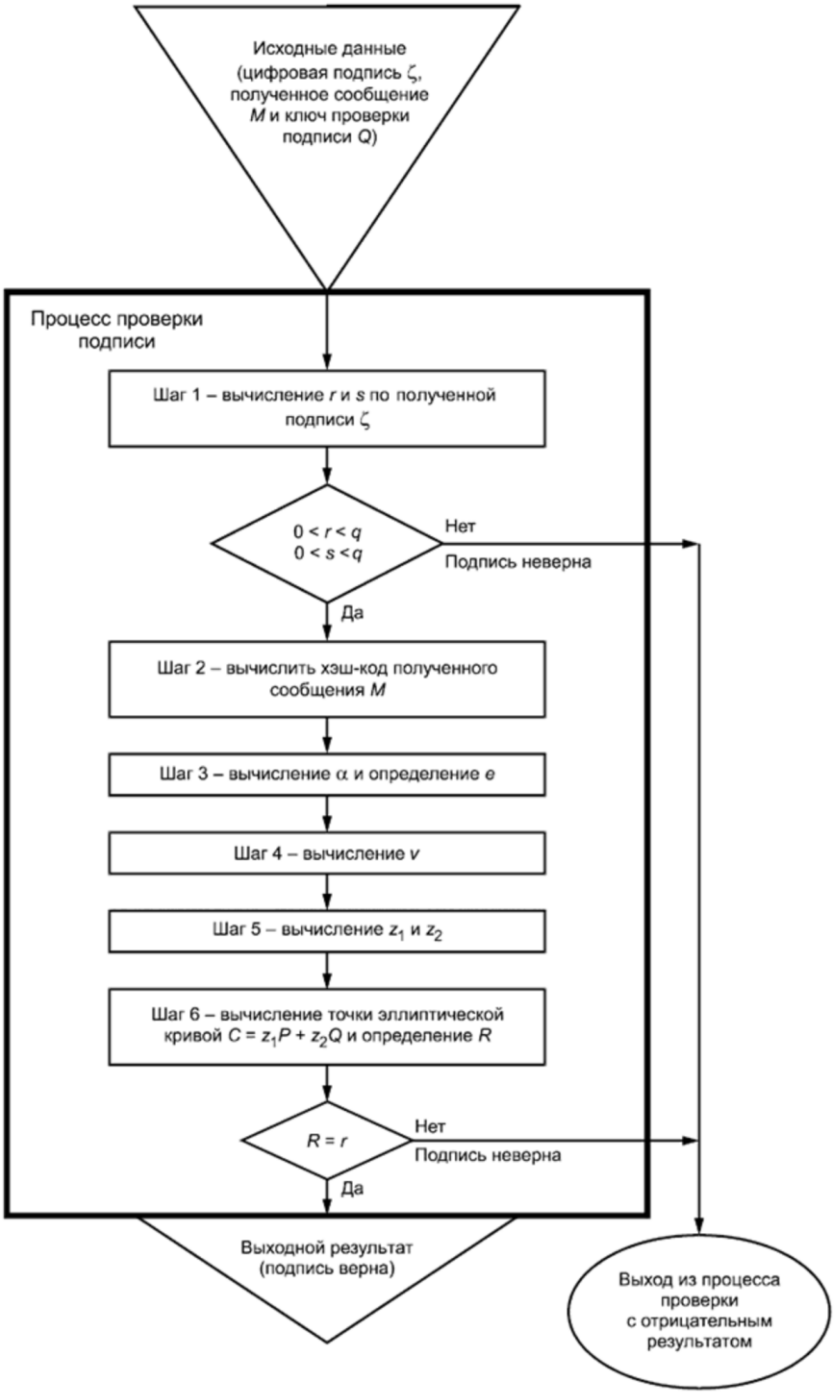
1. Изучить алгоритм цифровой подписи ГОСТ 3410.
2. Создать и протестировать алгоритм цифровой подписи ГОСТ 3410 на языке высокого уровня.

Ссылка на гит-хаб: https://github.com/ElephantT/MZI/tree/main/KR2

Блок - схема процесса формирования цифровой подписи



Блок - схема процесса проверки цифровой подписи



Код

Два файла - main.py, config.py. Лучше посмотреть код в самих файлах в архиве, но приложу и тут фотки исходного кода, без константных параметров, указанных в конфиге.

main.py

```
"""
Процесс формирования ЭЦП выполняется по следующему алгоритму:
Вычислить хеш сообщения M: H=h(M);
Вычислить целое число α, двоичным представлением которого является H;
Определить e=α mod q, если e=0, задать e=1;
Сгенерировать случайное число k, удовлетворяющее условию 0<k<q;
Вычислить точку эллиптической кривой C=k*P;
Определить r = xC mod q, где xC – x-координата точки C. Если r=0, то вернуться к шагу 4;
Вычислить значение s = (rd+ke) mod q. Если s=0, то вернуться к шагу 4;
Вернуть значение r||s в качестве цифровой подписи.
"""

from os import urandom
from hashlib import sha1
from codecs import getdecoder
from codecs import getencoder

from config import SIZE


def hex_encode(data):
    hex_encoder = getencoder('hex')
    return hex_encoder(data)[0].decode('ascii')


def hex_decode(data):
    hex_decoder = getdecoder('hex')
    return hex_decoder(data)[0]


def modinvert(a, n):
    if a < 0:
        return n - modinvert(-a, n)

    t, new_t = 0, 1
    r, new_r = n, a
    while new_r != 0:
        quotinent = r // new_r
        t, new_t = new_t, t - quotinent * new_t
        r, new_r = new_r, r - quotinent * new_r
    if r > 1:
        return -1
    if t < 0:
        t = t + n
    return t


def bytes2long(raw):
    return int(hex_encode(raw), 16)


def long2bytes(n, size=SIZE):
    res = hex(int(n))[2:].rstrip("L")

    if len(res) % 2 != 0:
        res = "0" + res

    s = hex_decode(res)

    if len(s) != size:
        s = (size - len(s)) * b"\x00" + s
```

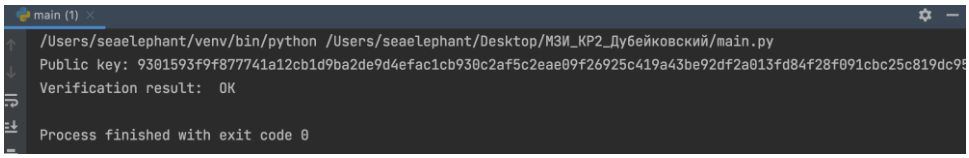
```
    if len(s) != size:
        s = (size - len(s)) * b"\x00" + s

    return s


class GOST3410Curve(object):
    def __init__(self, p, q, a, b, x, y, e=None, d=None):
        self.p = p
        self.q = q
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        self.e = e
        self.d = d

        r1 = self.y * self.y % self.p
        r2 = ((self.x * self.x + self.a) * self.x + self.b) % self.p

        if r1 != self.pos(r2):
            raise ValueError("Invalid parameters")

        self._st = None

    def pos(self, v):
        if v < 0:
            return v + self.p
        return v

    def _add(self, p1x, p1y, p2x, p2y):
        if p1x == p2x and p1y == p2y:
            t = ((3 * p1x * p1x + self.a) * modinvert(2 * p1y, self.p)) % self.p
        else:
            tx = self.pos(p2x - p1x) % self.p
            ty = self.pos(p2y - p1y) % self.p
            t = (ty * modinvert(tx, self.p)) % self.p

        tx = self.pos(t * t - p1x - p2x) % self.p
        ty = self.pos(t * (p1x - tx) - p1y) % self.p

        return tx, ty

    def exp(self, degree, x=None, y=None):
        x = x or self.x
        y = y or self.y
        tx = x
        ty = y
        if degree == 0:
            raise ValueError("Bad degree value")
        degree -= 1
        while degree != 0:
            if degree & 1 == 1:
                tx, ty = self._add(tx, ty, x, y)
            degree = degree >> 1
            x, y = self._add(x, y, x, y)
        return tx, ty
```

```
    def st(self):
        if self.e is None or self.d is None:
            raise ValueError("non twisted Edwards curve")
        if self._st is not None:
            return self._st
        self._st = (
            self.pos(self.e - self.d) * modinvert(4, self.p) % self.p,
            (self.e + self.d) * modinvert(6, self.p) % self.p,
        )
        return self._st


# Generates public key from the private one
def public_key(curve, prv):
    return curve.exp(prv)


# Calculates signature for provided digest
def sign(curve, prv, digest):
    size = SIZE * 2
    q = curve.q
    e = 1 if (bytes2long(digest) % q == 0) else bytes2long(digest) % q

    while True:
        k = bytes2long(urandom(size)) % q
        if k == 0:
            continue
        r, _ = curve.exp(k)
        r %= q
        if r == 0:
            continue
        d = prv * r
        k *= e
        s = (d + k) % q
        if s == 0:
            continue
        break
    return long2bytes(s, size) + long2bytes(r, size)

```

```
157
158     # Verifies provided digest with the signature
159     def verify(curve, pub, digest, signature):
160         size = SIZE * 2
161
162         if len(signature) != size * 2:
163             raise ValueError("Invalid signature length")
164
165         q = curve.q
166         p = curve.p
167         s = bytes2long(signature[:size])
168         r = bytes2long(signature[size:])
169
170         if r <= 0 or r >= q or s <= 0 or s >= q:
171             return False
172
173         e = bytes2long(digest) % curve.q
174
175         if e == 0:
176             e = 1
177         v = modinvert(e, q)
178         z1 = s * v % q
179         z2 = q - r * v % q
180         p1x, p1y = curve.exp(z1)
181         q1x, q1y = curve.exp(z2, pub[0], pub[1])
182         lm = q1x - p1x
183
184         if lm < 0:
185             lm += p
186         lm = modinvert(lm, p)
187         z1 = q1y - p1y
188         lm = lm * z1 % p
189         lm = lm * lm % p
190         lm = lm - p1x - q1x
191         lm = lm % p
192         if lm < 0:
193             lm += p
194         lm %= q
195         return lm == r
196
197
198     def prv_unmarshal(prv):
199         return bytes2long(prv[::-1])
200
201
202     def pub_marshal(pub):
203         return (long2bytes(pub[1], SIZE) + long2bytes(pub[0], SIZE))[::-1]
204
205
206     def pub_unmarshal(pub):
207         pub = pub[::-1]
208         return bytes2long(pub[SIZE:]), bytes2long(pub[:SIZE])
```

```
205
206  def pub_unmarshal(pub):
207      pub = pub[::-1]
208      return bytes2long(pub[SIZE:]), bytes2long(pub[:SIZE])
209
210
211  # Converts Edwards curve U,V coordinates to Weierstrass X,Y
212  def uv2xy(curve, u, v):
213      s, t = curve.st()
214      k1 = (s * (1 + v)) % curve.p
215      k2 = curve.pos(1 - v)
216      x = t + k1 * modinvert(k2, curve.p)
217      y = k1 * modinvert(u * k2, curve.p)
218      return x % curve.p, y % curve.p
219
220
221  # Converts Weierstrass X, Y coordinates to Edwards curve U, V
222  def xy2uv(curve, x, y):
223      s, t = curve.st()
224      xmt = curve.pos(x - t)
225      u = xmt * modinvert(y, curve.p)
226      v = curve.pos(xmt - s) * modinvert(xmt + s, curve.p)
227      return u % curve.p, v % curve.p
228
229
230  def main():
231      curve = GOST3410Curve(
232          p=bytes2long(hex_decode("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97")),
233          q=bytes2long(hex_decode("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF6C611070995AD10045841B09B761B893")),
234          a=bytes2long(hex_decode("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD94")),
235          b=bytes2long(hex_decode("00000000000000000000000000000000000000000000000000000000000000a6")),
236          x=bytes2long(hex_decode("0000000000000000000000000000000000000000000000000000000000000001")),
237          y=bytes2long(hex_decode("8D91E471E0989CDA27DF505A453F2B7635294F2DDF23E3B122ACC99C9E9F1E14")),
238      )
239
240      prv_raw = urandom(32)
241      prv = prv_unmarshal(prv_raw)
242      pub = public_key(curve, prv)
243      print("Public key:", hex_encode(pub_marshal(pub)))
244      data_for_signing = b"some data"
245      dgst = sha1(data_for_signing).digest()
246      signature = sign(curve, prv, dgst)
247
248      res = verify(curve, pub, dgst, signature)
249      print("Verification result: ", 'OK' if res else 'NOT OK')
250
251
252  if __name__ == '__main__':
253      main()
```

Пример выполнения



/Users/seaelephant/venv/bin/python
/Users/seaelephant/Desktop/МЗИ_КР2_Дубейковский/main.py

Public key:
9301593f9f877741a12cb1d9ba2de9d4efac1cb930c2af5c2eae09f26925c419a43be92df2a013fd
84f28f091cbc25c819dc95ca1206a5f407fc680b91576cd5
Verification result: OK

Process finished with exit code 0