БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСиС

Специальность ИиТП

Контрольная работа № 2

по дисциплине «Методы защиты информации»

Выполнил: Реут М.В.

группа 893551

Минск 2022

# **1. Постановка задачи**

1. Изучить алгоритм цифровой подписи ГОСТ 3410.

2. Создать и протестировать алгоритм цифровой подписи ГОСТ 3410 на

языке высокого уровня.

# **2. Общие положения**

Общепризнанная схема (модель) цифровой подписи охватывает следующие процессы:

* генерация ключей (подписи и проверки подписи);
* формирование подписи;
* проверка подписи.

Механизм цифровой подписи определяется посредством реализации двух

основных процессов (см. раздел 4):

– формирование подписи (см. 4.1);

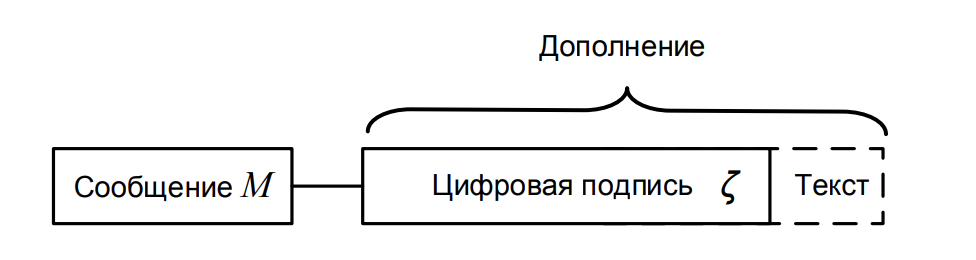
– проверка подписи (см. 4.2).

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.

Схематическое представление подписанного сообщения показано на рисунок 1.



*Рисунок 1 – Схема подписанного сообщения*

Поле «Текст», показанное на данном рисунке и дополняющее поле «Цифровая подпись», может, например, содержать идентификаторы субъекта, подписавшего сообщение, и/или метку времени.

Установленная в настоящем стандарте схема цифровой подписи должна быть реализована с использованием операций группы точек эллиптической кривой, определённой над конечным простым полем, а также хэш–функции.

Криптографическая стойкость данной схемы цифровой подписи основывается на

сложности решения задачи дискретного логарифмирования в группе точек

эллиптической кривой, а также на стойкости используемой хэш–функции. Алгоритмы вычисления хэш–функции установлены в ГОСТ Р 34.11-2012.

ГОСТ Р 34.10-2012

Параметры схемы цифровой подписи, необходимые для ее формирования и проверки, определены в 3.2. В настоящем стандарте предусмотрена возможность выбора одного из двух вариантов требований к параметрам.

Настоящий стандарт не определяет процесс генерации параметров схемы

цифровой подписи. Конкретный алгоритм (способ) реализации данного процесса определяется субъектами схемы цифровой подписи исходя из требований к аппаратно-программным средствам, реализующим электронный документооборот.

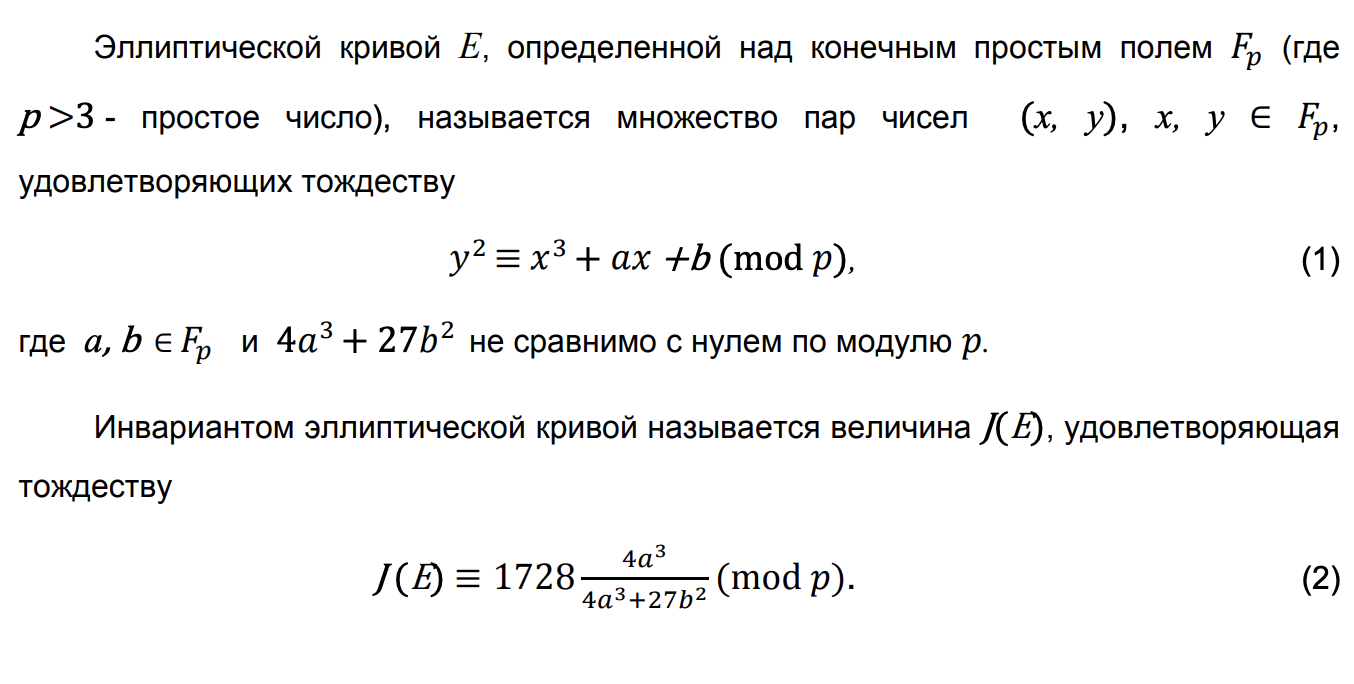
Цифровая подпись, представленная в виде двоичного вектора длиной 512 или 1024 бита, должна вычисляться с помощью определенного набора правил, изложенных в 4.1.

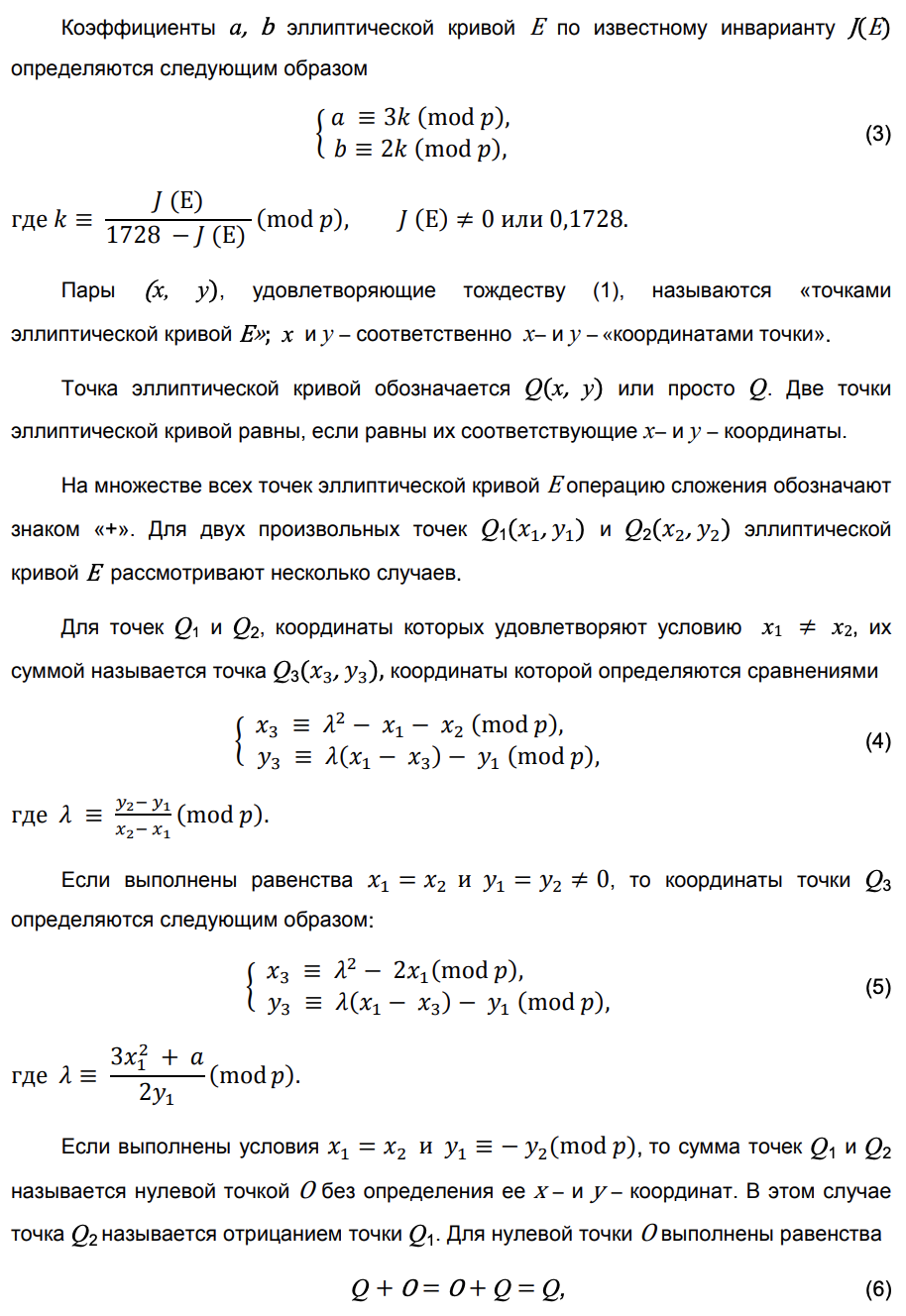
Набор правил, позволяющих принять либо отвергнуть цифровую подпись под полученным сообщением, установлен в 4.2.

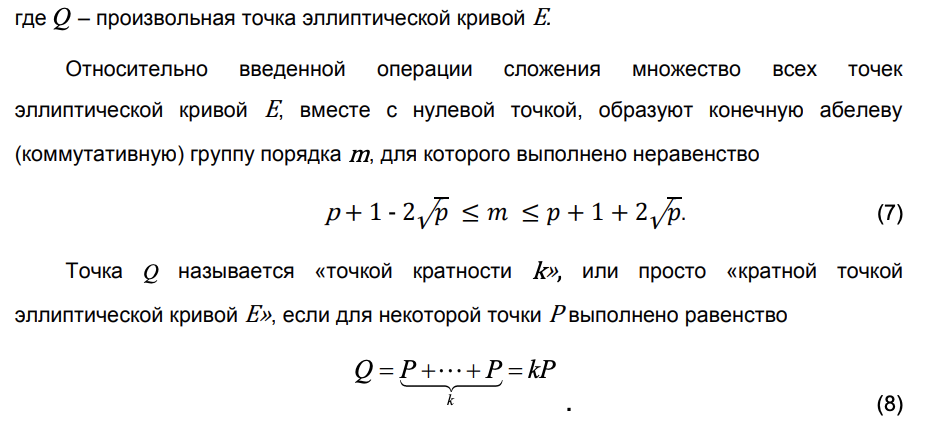
# **3. Математические объекты**

Для определения схемы цифровой подписи необходимо описать базовые математические объекты, используемые в процессах ее формирования и проверки. В данном разделе установлены основные математические определения и требования, предъявляемые к параметрам схемы цифровой подписи.

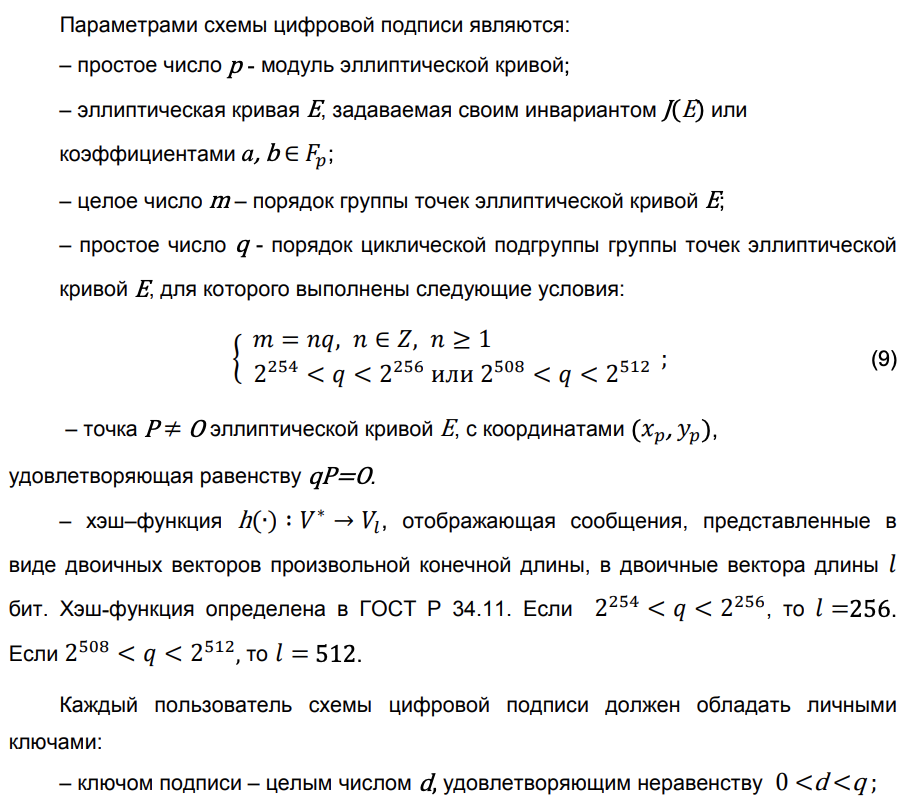
## 3.1. Математические определения

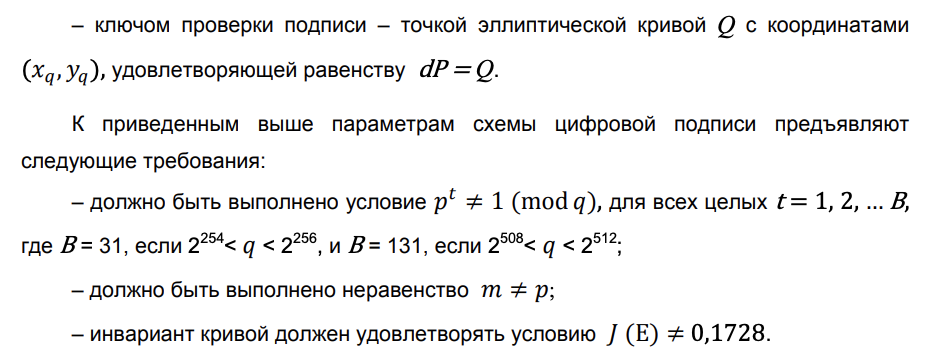




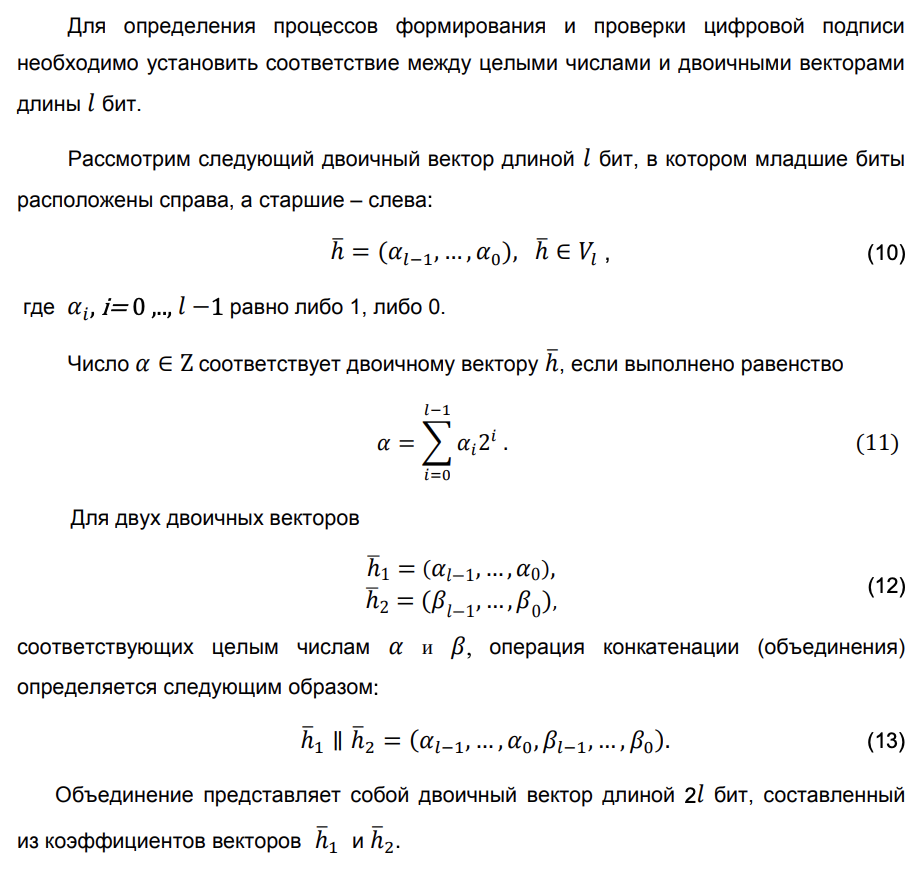


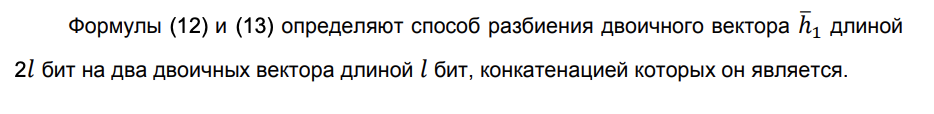
## 3.2. Параметры цифровой подписи





## 3.3. Двоичные векторы

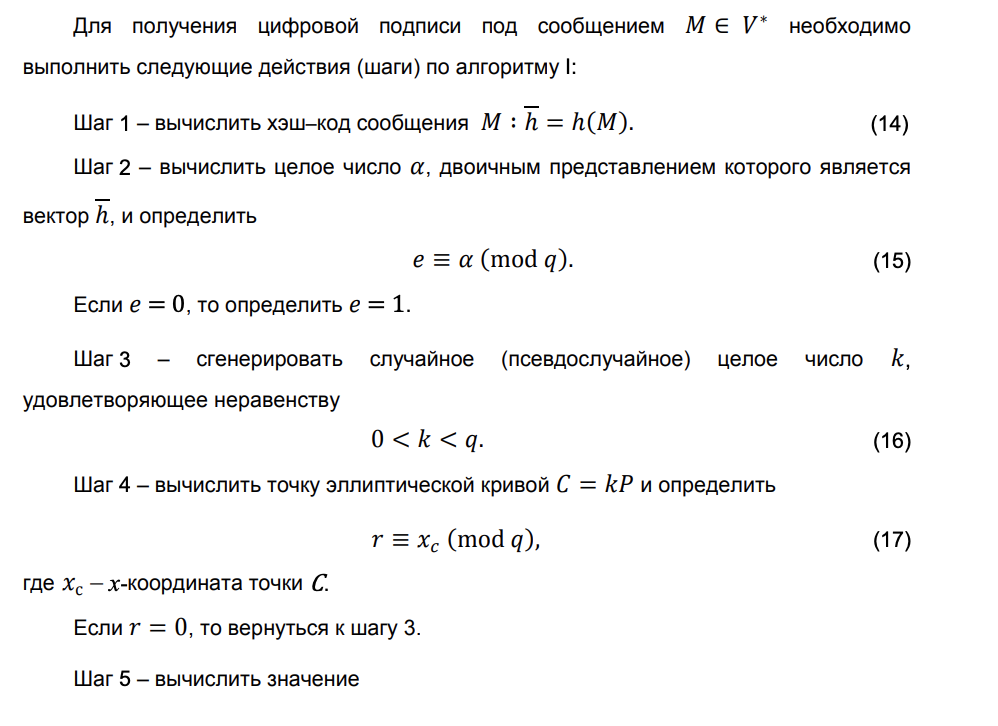


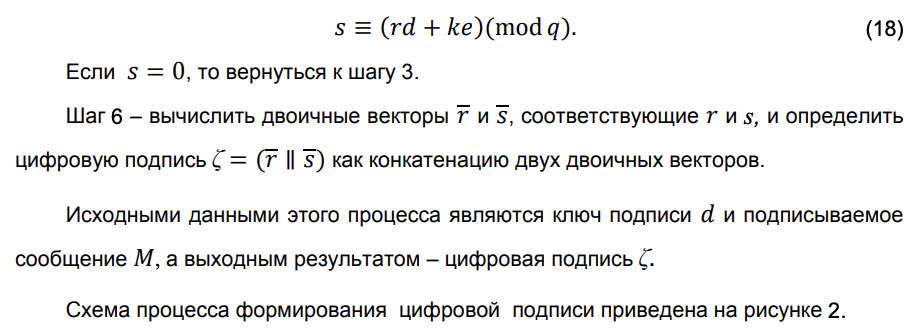


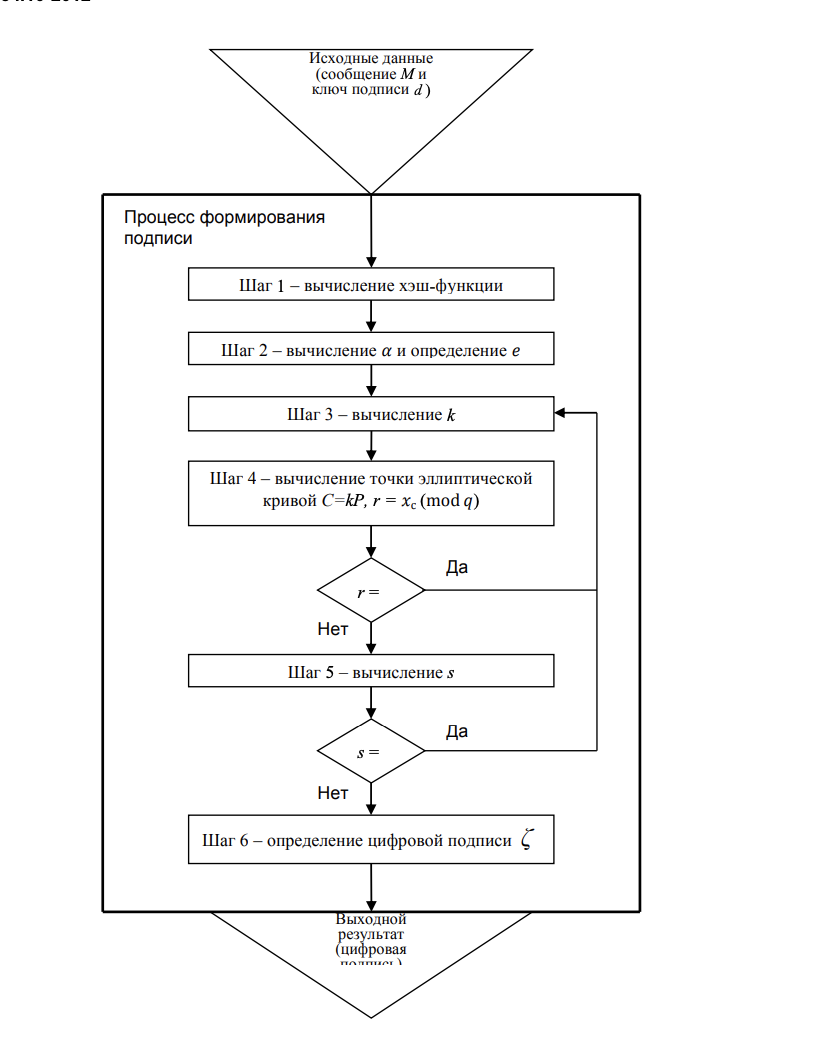
# **4. Основные процессы**

В данном разделе определены процессы формирования и проверки цифровой подписи под сообщением пользователя. Для реализации данных процессов необходимо, чтобы всем пользователям были известны параметры схемы цифровой подписи, соответствующие требованиям 3.2. Кроме того, каждый пользователь должен иметь ключ подписи d и ключ проверки подписи Q( , ), которые также должны соответствовать требованиям 3.2.

## 4.1. Формирование цифровой подписи



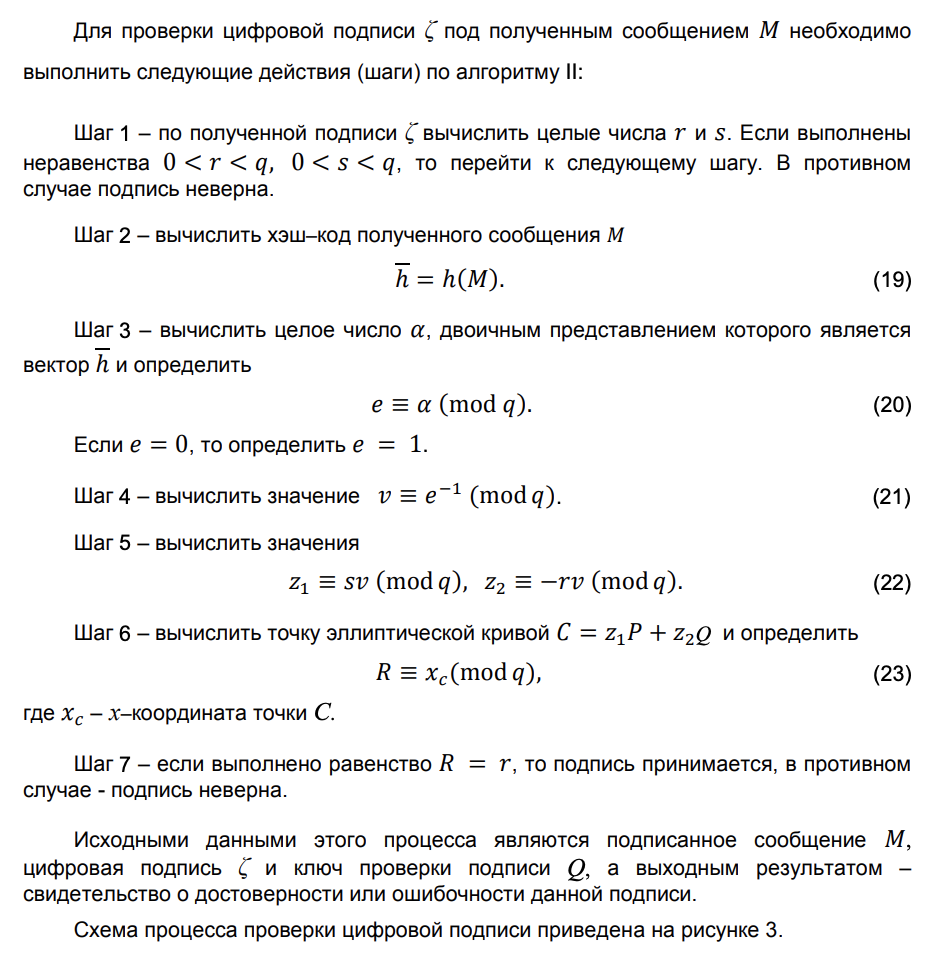


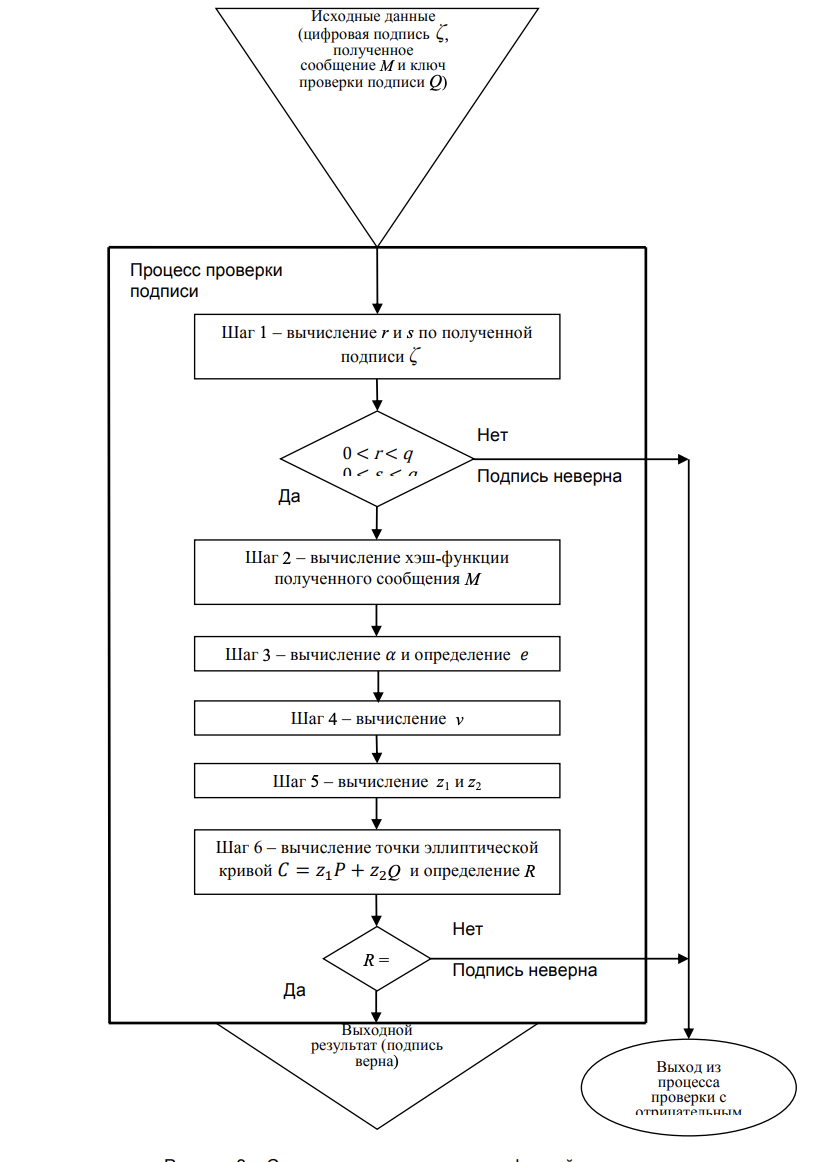


*Рисунок 2 – Схема процесса формирования цифровой подписи*

## 

## 4.2. Проверка цифровой подписи



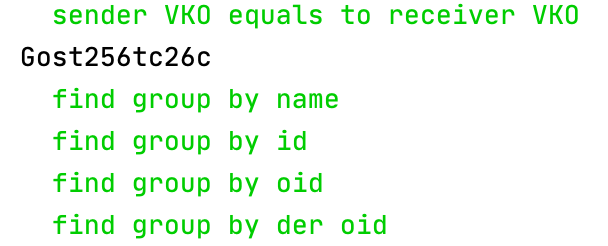
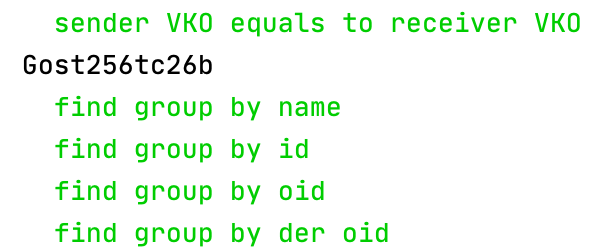
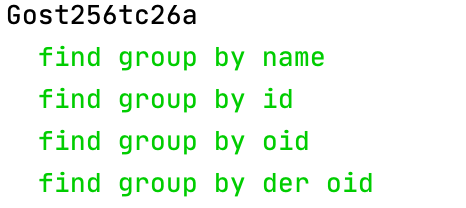


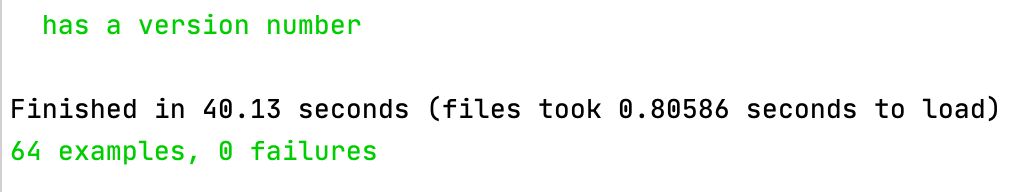
*Рисунок 3 – Схема процесса проверки цифровой подписи*

# **5. Скриншоты ввода данных и результатов выполнения программы**

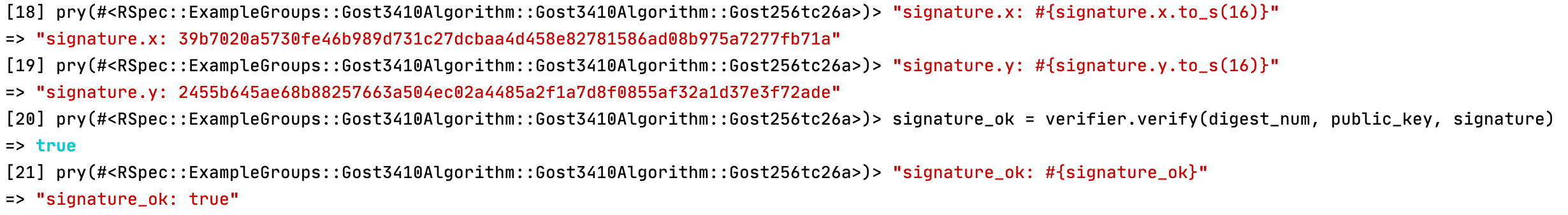
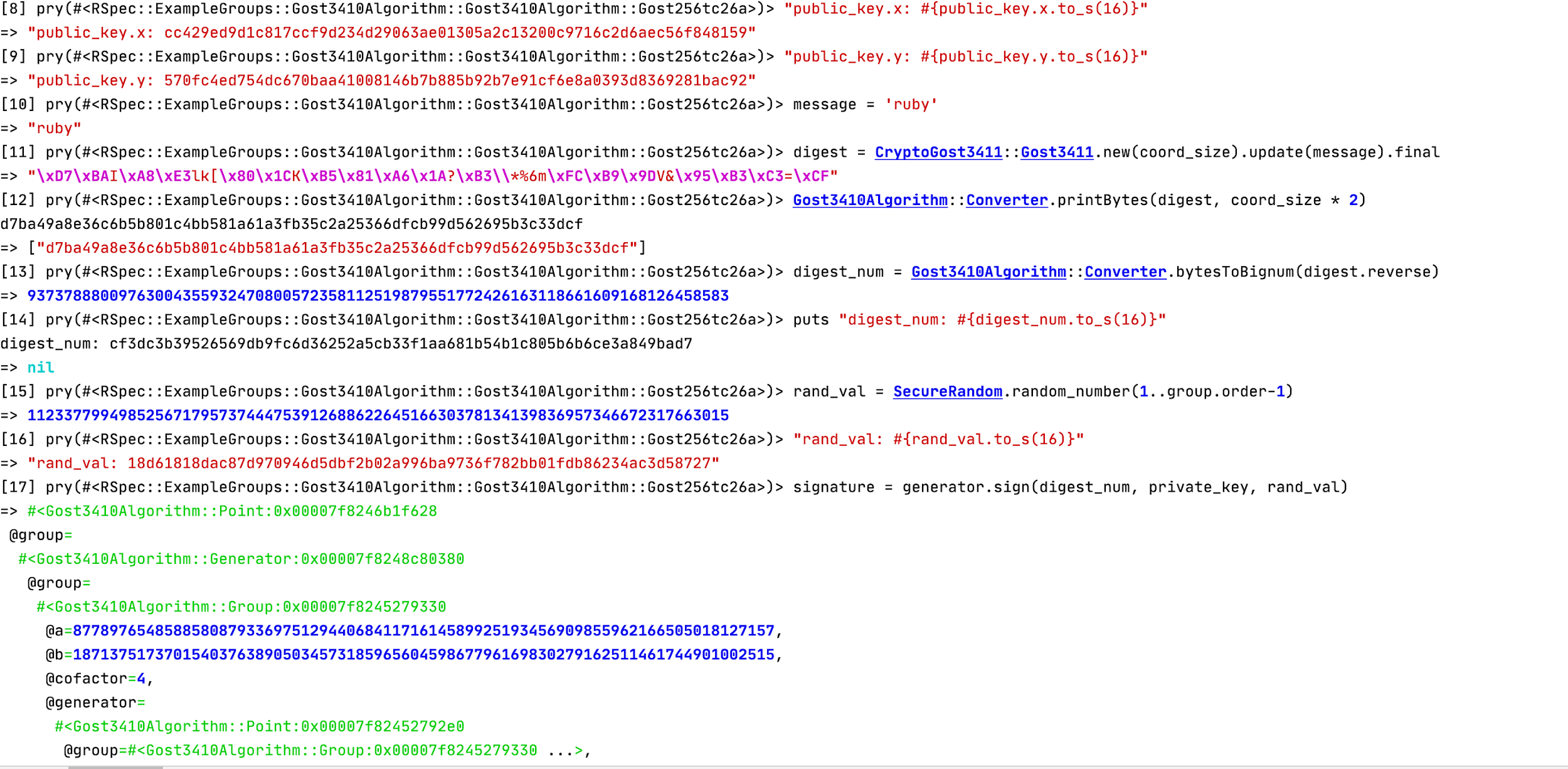
Чтобы упростить демонстрацию работы программы и его проверку в папке **spec** добавлены тесты.

Запуск тестов осуществляется при помощи команды ***rspec***.





Мы можем вклинится в этот процесс при помощи **“binding.pry”**



Полный алгоритм использования указан в **README.md**

# **6. Исходные файлы:**

## Реализация алгоритма:

**gost3410\_algorithm.rb**

*# Gost cryptography*

module *Gost3410Algorithm*

*require* 'securerandom'

*require* 'gost3410\_algorithm/version'

*require* 'gost3410\_algorithm/generator'

*require* 'gost3410\_algorithm/verifier'

*require* 'gost3410\_algorithm/point'

*require* 'gost3410\_algorithm/group'

*require* 'gost3410\_algorithm/modular\_arithmetic'

*require* 'gost3410\_algorithm/converter'

end

**converter.rb**

module *Gost3410Algorithm*

class *Converter*

*# Big number -> byte string (big-endian) of given size*

def self.*bignumToBytes*(*bn*, *size*)

*s* = *bn*.to\_s(16)

*s* = ('0' \* (*size* \* 2 - *s*.length)) + *s*

*sa* = *s*.scan(**/**..**/**)

*ba* = []

*sa*.each{|*c*| *ba* << *c*.to\_i(16).chr}

*bytes* = *ba*.join

end

*# Byte string (big-endian) to big number*

def self.*bytesToBignum*(*bytes*)

*bytes*.unpack('H\*')[0].hex

end

*# Byte string hex printer*

def self.*printBytes*(*bytes*, *line\_size* = 16)

*bytes*.unpack('H\*')[0].scan(**/**.{1,#{*line\_size*}}**/**).each{|*s*| puts(*s*)}

end

end

end

**generator.rb**

module *Gost3410Algorithm*

*# DigitalSignature*

*#*

class *Generator*

*attr\_reader* :group

def *initialize*(*group*)

@group = *group*

end

def *sign*(*hash*, *private\_key*, *rand\_val*)

@hash = *hash*

@private\_key = *private\_key*

@rnd = *rand\_val*

@r = r\_func

*s* = s\_func

*Gost3410Algorithm*::*Point*.new self, [@r, *s*]

end

def *vko*(*ukm*, *private\_key*, *other\_public\_key*)

*n* = (group.opts[:h] \* *private\_key* \* *ukm*) % group.order

*other\_public\_key* \* *n*

end

*private*

def *r\_func*

(group.generator \* @rnd).x % group.order

end

def *s\_func*

(@r \* @private\_key + @rnd \* @hash) % group.order

end

end

end

**group.rb**

module *Gost3410Algorithm*

*# Group*

class *Group*

*attr\_reader* :opts, :generator, :a, :b, :gx, :gy, :order, :p

def *initialize*(*opts*)

@opts = *opts*

@name = *opts*.fetch(:name)

@p = *opts*[:p]

@a = *opts*[:a]

@b = *opts*[:b]

@gx = *opts*[:gx]

@gy = *opts*[:gy]

@order = *opts*[:n]

@cofactor = *opts*[:h]

@generator = *Gost3410Algorithm*::*Point*.new self, [gx, gy]

end

*NAMES* = %w[

Gost256tc26test

Gost256tc26a

Gost256tc26b

Gost256tc26c

Gost256tc26d

Gost512tc26test

Gost512tc26a

Gost512tc26b

Gost512tc26c

].freeze

*NAMES*.each do |*name*|

*require\_relative* "./group/#{*name*.downcase}"

end

def self.*findByOid*(*oid*)

*group* = nil

*NAMES*.each do |*n*|

*g* = *Object*.const\_get("Gost3410Algorithm::Group::#{*n*}")

if *g*.opts[:oid] == *oid* then

*group* = *g*

break

end

if (*g*.opts[:cp\_oid] != nil) && (*g*.opts[:cp\_oid] == *oid*) then

*group* = *g*

break

end

end

*group*

end

def self.*findByDerOid*(*der\_oid*)

*group* = nil

*NAMES*.each do |*n*|

*g* = *Object*.const\_get("Gost3410Algorithm::Group::#{*n*}")

if *g*.opts[:der\_oid] == *der\_oid* then

*group* = *g*

break

end

if (*g*.opts[:cp\_der\_oid] != nil) && (*g*.opts[:cp\_der\_oid] == *der\_oid*) then

*group* = *g*

break

end

end

*group*

end

def self.*findById*(*id*)

*group* = nil

*NAMES*.each do |*n*|

*g* = *Object*.const\_get("Gost3410Algorithm::Group::#{*n*}")

if *g*.opts[:id] == *id* then

*group* = *g*

break

end

if (*g*.opts[:cp\_id] != nil) && (*g*.opts[:cp\_id] == *id*) then

*group* = *g*

break

end

end

*group*

end

def self.*findByName*(*name*)

*group* = nil

*NAMES*.each do |*n*|

*g* = *Object*.const\_get("Gost3410Algorithm::Group::#{*n*}")

if *g*.opts[:name] == *name* then

*group* = *g*

break

end

end

*group*

end

def *generate\_public\_key*(*private\_key*)

generator \* *private\_key*

end

end

end

**modular\_arithmetic.rb**

module *Gost3410Algorithm*

*# ModularArithmetic*

module *ModularArithmetic*

*module\_function*

def *gcd*(*x*, *y*)

gcdext(*x*, *y*).first

end

def *gcdext*(*x*, *y*)

if *x* < 0

*g*, *a*, *b* = gcdext(-*x*, *y*)

return [*g*, -*a*, *b*]

end

if *y* < 0

*g*, *a*, *b* = gcdext(*x*, -*y*)

return [*g*, *a*, -*b*]

end

*r0*, *r1* = *x*, *y*

*a0* = *b1* = 1

*a1* = *b0* = 0

until *r1*.zero?

*q* = *r0* / *r1*

*r0*, *r1* = *r1*, *r0* - *q*\**r1*

*a0*, *a1* = *a1*, *a0* - *q*\**a1*

*b0*, *b1* = *b1*, *b0* - *q*\**b1*

end

[*r0*, *a0*, *b0*]

end

def *invert*(*num*, *mod*)

*g*, *a*, *b* = gcdext(*num*, *mod*)

unless *g* == 1

*raise ZeroDivisionError*.new("#{*num*} has no inverse modulo #{*mod*}")

end

*a* % *mod*

end

end

end

**point.rb**

module *Gost3410Algorithm*

*# EllipticCurvePoint*

class *Point*

*attr\_accessor* :x, :y, :group

def *initialize*(*group*, *coords*)

@group = *group*

@x, @y = *coords*

end

def *coords*

[x, y]

end

*# rubocop:disable Metrics/AbcSize*

def *+*(*other*)

unless *other*.is\_a? *Point*

*raise ArgumentError*, "Invalid other: #{*other*.inspect}"

end

*new\_x* = add\_ec\_module(*other*.x - x)

*new\_y* = add\_ec\_module(*other*.y - y)

*s* = add\_ec\_module(

(*new\_y* \* *ModularArithmetic*.invert(*new\_x*, group.p)) % group.p

)

*new\_x* = add\_ec\_module((*s*\*\*2 - x - *other*.x) % group.p)

*new\_y* = add\_ec\_module((*s* \* (x - *new\_x*) - y) % group.p)

self.class.new group, [*new\_x*, *new\_y*]

end

def *double*

*new\_x* = add\_ec\_module(2 \* y)

*new\_y* = add\_ec\_module(3 \* x\*\*2 + group.a)

*s* = (*new\_y* \* *ModularArithmetic*.invert(*new\_x*, group.p)) % group.p

*new\_x* = add\_ec\_module(*s*\*\*2 - 2 \* x) % group.p

*new\_y* = add\_ec\_module(*s* \* (x - *new\_x*) - y) % group.p

self.class.new group, [*new\_x*, *new\_y*]

end

*# rubocop:enable Metrics/AbcSize*

def *\**(*other*)

return unless *other*.is\_a? *Numeric*

if *other* == 1

self

elsif (*other* % 2).odd?

self + (self \* (*other* - 1))

else

double \* (*other* / 2)

end

end

def *add\_ec\_module*(*coord*)

*coord* < 0 ? *coord* + group.p : *coord*

end

end

end

**verifier.rb**

module *Gost3410Algorithm*

*# DigitalSignature*

class *Verifier*

*attr\_reader* :group

def *initialize*(*group*)

@group = *group*

end

def *verify*(*hash*, *public\_key*, *signature*)

@public\_key = *public\_key*

*r* = *signature*.x

*s* = *signature*.y

return false if invalid\_vector?(*r*) || invalid\_vector?(*s*)

(c\_param(*hash*, *public\_key*, *r*, *s*).x % group.order) == *r*

end

*private*

def *mod\_inv*(*opt*, *mod*)

*ModularArithmetic*.invert(*opt*, *mod*)

end

def *valid\_vector?*(*vector*)

(1...group.order).cover? *vector*

end

def *invalid\_vector?*(*vector*)

!valid\_vector?(*vector*)

end

def *z\_param*(*hash*, *param*)

(*param* \* mod\_inv(*hash*, group.order)) % group.order

end

def *c\_param*(*hash*, *public\_key*, *r*, *s*)

group.generator \* z\_param(*hash*, *s*) + *public\_key* \* z\_param(*hash*, -*r*)

end

end

end

## Тестирование алгоритма:

**gost3410\_algorithm\_spec.rb**

*describe Gost3410Algorithm* do

*context* 'gost3410\_algorithm' do

*NAMES* = %w[

Gost256tc26test

Gost256tc26a

Gost256tc26b

Gost256tc26c

Gost256tc26d

Gost512tc26test

Gost512tc26a

Gost512tc26b

Gost512tc26c

].freeze

*NAMES*.each do |*name*|

*context name* do

*let*(:group) { *Object*.const\_get("Gost3410Algorithm::Group::#{*name*}") }

*let*(:private\_key) { *SecureRandom*.random\_number(1..group.order-1) }

*let*(:public\_key) { group.generate\_public\_key private\_key }

*let*(:message) { *Faker*::*Lorem*.sentence(3) }

*let*(:coord\_size) { group.opts[:coord\_size] }

*let*(:digest) { *CryptoGost3411*::*Gost3411*.new(coord\_size).update(message).final }

*let*(:digest\_num) { *Gost3410Algorithm*::*Converter*.bytesToBignum(digest.reverse) }

*let*(:generator) { *Gost3410Algorithm*::*Generator*.new(group) }

*let*(:rand\_val) { *SecureRandom*.random\_number(1..group.order-1) }

*let*(:signature) { generator.sign(digest\_num, private\_key, rand\_val) }

*let*(:verifier) { *Gost3410Algorithm*::*Verifier*.new(group) }

*let*(:another\_message) { *Faker*::*Lorem*.sentence(2) }

*let*(:another\_digest) { *CryptoGost3411*::*Gost3411*.new(coord\_size).update(another\_message).final}

*let*(:another\_digest\_num) { *Gost3410Algorithm*::*Converter*.bytesToBignum(another\_digest.reverse) }

*let*(:receiver\_private\_key) { *SecureRandom*.random\_number(1..group.order-1) }

*let*(:receiver\_public\_key) { group.generate\_public\_key receiver\_private\_key }

*let*(:ukm) { *SecureRandom*.random\_number(2\*\*(coord\_size\*2)..2\*\*(coord\_size\*8)-1) }

*let*(:sender\_vko) { generator.vko(ukm, private\_key, receiver\_public\_key) }

*let*(:receiver\_vko) { generator.vko(ukm, receiver\_private\_key, public\_key) }

*it* 'find group by name' do

expect(*Gost3410Algorithm*::*Group*.findByName(group.opts[:name]) == group).to be\_truthy

end

*it* 'find group by id' do

expect(*Gost3410Algorithm*::*Group*.findById(group.opts[:id]) == group).to be\_truthy

end

*it* 'find group by oid' do

expect(*Gost3410Algorithm*::*Group*.findByOid(group.opts[:oid]) == group).to be\_truthy

end

*it* 'find group by der oid' do

expect(*Gost3410Algorithm*::*Group*.findByDerOid(group.opts[:der\_oid]) == group).to be\_truthy

end

*it* 'has valid signature' do

expect(verifier.*verify*(digest\_num, public\_key, signature)).to be\_truthy

end

*it* 'has invalid signature for changed message' do

expect(verifier.*verify*(another\_digest\_num, public\_key, signature)).to be\_falsy

end

*it* 'sender VKO equals to receiver VKO' do

expect((sender\_vko.x == receiver\_vko.x) && (sender\_vko.y == receiver\_vko.y)).to be\_truthy

end

end

end

end

end

## Использование алгоритма.

## **README.md**

Ruby implementing GOST 34.10-2012 ECC signature and VKO algorithms.

*# Installation*

*$ bundle*

*# Usage*

group = Gost3410Algorithm::Group::Gost256tc26a

puts group.opts[:name]

puts group.opts[:id]

puts group.opts[:oid]

coord\_size = group.opts[:coord\_size] # in bytes

generator = Gost3410Algorithm::Generator.new(group) # for sign and for vko

verifier = Gost3410Algorithm::Verifier.new(group) # for verify

private\_key is random number 0<n<group.order

you can use fixed number for debugging

private\_key = SecureRandom.random\_number(1..group.order-1)

puts "private\_key: #{private\_key.to\_s(16)}"

public\_key = group.generate\_public\_key private\_key

puts "public\_key.x: #{public\_key.x.to\_s(16)}"

puts "public\_key.y: #{public\_key.y.to\_s(16)}"

*# Signature:*

message = 'ruby'

digest = CryptoGost3411::Gost3411.new(coord\_size).update(message).final

Gost3410Algorithm::Converter.printBytes(digest, coord\_size \* 2)

digest\_num = Gost3410Algorithm::Converter.bytesToBignum(digest.reverse)

puts "digest\_num: #{digest\_num.to\_s(16)}"

rnd\_val is random number 0<n<group.order so as private\_key

you can use fixed number for debugging

rand\_val = SecureRandom.random\_number(1..group.order-1)

puts "rand\_val: #{rand\_val.to\_s(16)}"

signature = generator.sign(digest\_num, private\_key, rand\_val)

puts "signature.x: #{signature.x.to\_s(16)}" # x stands for r

puts "signature.y: #{signature.y.to\_s(16)}" # y stands for s

signature\_ok = verifier.verify(digest\_num, public\_key, signature)

puts "signature\_ok: #{signature\_ok}"

*# VKO:*

receiver\_private\_key = SecureRandom.random\_number(1..group.order-1)

puts "receiver\_private\_key: #{receiver\_private\_key.to\_s(16)}"

receiver\_public\_key = group.generate\_public\_key receiver\_private\_key

puts "receiver\_public\_key.x: #{receiver\_public\_key.x.to\_s(16)}"

puts "receiver\_public\_key.y: #{receiver\_public\_key.y.to\_s(16)}"

ukm is random number 2\*\*(coord\_size\*2)<=ukm<2\*\*(coord\_size\*8)

ukm = SecureRandom.random\_number(2\*\*(coord\_size\*2)..2\*\*(coord\_size\*8)-1)

puts "ukm: #{ukm.to\_s(16)}"

sender\_vko = generator.vko(ukm, private\_key, receiver\_public\_key)

puts "sender\_vko.x: #{sender\_vko.x.to\_s(16)}"

puts "sender\_vko.y: #{sender\_vko.y.to\_s(16)}"

receiver\_vko = generator.vko(ukm, receiver\_private\_key, public\_key)

puts "receiver\_vko.x: #{receiver\_vko.x.to\_s(16)}"

puts "receiver\_vko.y: #{receiver\_vko.y.to\_s(16)}"

puts "vko\_ok: #{(sender\_vko.x == receiver\_vko.x) && (sender\_vko.y == receiver\_vko.y)}"

*#### Converting GOST digest, public\_key and signature strings to big numbers and vise versa*

group = Gost3410Algorithm::Group::Gost256tc26test

coord\_size = group.opts[:coord\_size] # in bytes

get tbs, signer public\_key and signature byte strings from x509 certificate

generate digest

digest32 = CryptoGost3411::Gost3411.new(coord\_size).update(tbs).final

little-endian digest32 -> reverse -> big-endian -> BigNum

digest32\_num = Gost3410Algorithm::Converter.bytesToBignum(digest32.reverse)

public key x||y little-endian -> reverse -> big-endian -> Point(x\_num, y\_num)

public\_key\_x\_num = Gost3410Algorithm::Converter.bytesToBignum(public\_key[0...coord\_size].reverse)

public\_key\_y\_num = Gost3410Algorithm::Converter.bytesToBignum(public\_key[coord\_size..-1].reverse)

public\_key\_point = Gost3410Algorithm::Point.new(group, [public\_key\_x\_num, public\_key\_y\_num])

signature s||r big-endian -> Point(r\_num, s\_num)

signature\_s\_num = Gost3410Algorithm::Converter.bytesToBignum(signature[0...coord\_size])

signature\_r\_num = Gost3410Algorithm::Converter.bytesToBignum(signature[coord\_size..-1])

signature\_point = Gost3410Algorithm::Point.new(group, [sig\_r\_num, sig\_s\_num])

signature\_point\_ok = Gost3410Algorithm::Verifier.new(group).verify(digest32\_num, public\_key\_point, signature\_point)

puts "signature\_point\_ok: #{signature\_point\_ok}"

*#### Converting signature point to byte string ( s||r, big-endian):*

signature\_bytes\_s = Gost3410Algorithm::Converter.bignumToBytes(signature\_point.x, coord\_size)

signature\_bytes\_r = Gost3410Algorithm::Converter.bignumToBytes(signature\_point.y, coord\_size)

signature\_bytes = signature\_bytes\_s + signature\_bytes\_r

*#### Print signature bytes in hex (64 symbols in line)*

Gost3410Algorithm::Converter.printBytes(signature\_bytes, coord\_size \* 2)

*# List of TC 26 GOST 34.10-2012 elliptic curves:*

Gost256tc26test - id-tc26-gost-3410-2012-256-paramSetTest (former id-GostR3410-2001-TestParamSet). Use for testing only!

Gost256tc26a - id-tc26-gost-3410-2012-256-paramSetA (Edwards twisted curve)

Gost256tc26b - id-tc26-gost-3410-2012-256-paramSetB (former id-GostR3410-2001-CryptoPro-A-ParamSet)

Gost256tc26c - id-tc26-gost-3410-2012-256-paramSetC (former id-GostR3410-2001-CryptoPro-B-ParamSet)

Gost256tc26d - id-tc26-gost-3410-2012-256-paramSetD (former id-GostR3410-2001-CryptoPro-C-ParamSet)

Gost512test - id-tc26-gost-3410-2012-512-paramSetTest. Use for testing only!

Gost512tc26a - id-tc26-gost-3410-2012-512-paramSetA

Gost512tc26b - id-tc26-gost-3410-2012-512-paramSetB

Gost512tc26c - id-tc26-gost-3410-2012-512-paramSetC (Edwards twisted curve)

You can find group by name, by id, by oid and by der\_oid:

name = 'Gost256tc26test'

group = Gost3410Algorithm::Group.findByName(name)

id = 'id-tc26-gost-3410-2012-256-paramSetTest'

group = Gost3410Algorithm::Group.findById(id)

oid = '1.2.643.7.1.2.1.1.0'

group = Gost3410Algorithm::Group.findByOid(oid)

der\_oid = "\x06\x09\x2a\x85\x03\x07\x01\x02\x01\x01\x00"

group = Gost3410Algorithm::Group.findByDerOid(der\_oid)

*# Testing*

*$ rspec*