Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №7

Криптография с использованием эллиптических кривых

Выполнил:

Зеневич А. В.

Проверил:

Артемьев В.С.

Минск 2019

Содержание

[1. Постановка задачиError: Reference source not found](#_Toc506485975)2

2. Краткие теоретические сведения

3[. Результаты выполненияPAGEREF \_Toc506485977 \hError: Reference source not found](#_Toc506485977)

4[. ВыводыPAGEREF \_Toc506485978 \hError: Reference source not found](#_Toc506485978)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.Error: Reference source not foundError: Reference source not foundPAGEREF \_Toc506485979 \hError: Reference source not found](#_Toc506485979)

# 1. Постановка задачи

1. Изучить теоретические сведения.

2. Реализовать схему шифрования (дешифрования) для аналога алгоритма Диффи-Хеллмана на основе эллиптических кривых.

**2. Краткие теоретические сведения**

**Эллиптическая криптография** — раздел [криптографии](https://ru.wikipedia.org/wiki/Криптография), который изучает [асимметричные криптосистемы](https://ru.wikipedia.org/wiki/Криптосистема_с_открытым_ключом), основанные на [эллиптических кривых](https://ru.wikipedia.org/wiki/Эллиптическая_кривая) над [конечными полями](https://ru.wikipedia.org/wiki/Конечное_поле). Основное преимущество эллиптической криптографии заключается в том, что на сегодняшний день не известно существование субэкспоненциальных [алгоритмов](https://ru.wikipedia.org/wiki/Алгоритмы) решения задачи [дискретного логарифмирования](https://ru.wikipedia.org/wiki/Дискретное_логарифмирование).

## Описание алгоритма Деффи-Хеллмана:

При работе алгоритма каждая сторона:

1. генерирует случайное [натуральное число](https://ru.wikipedia.org/wiki/Натуральное_число) *a* — *закрытый ключ*
2. совместно с удалённой стороной устанавливает *открытые параметры* *p* и *g* (обычно значения *p* и *g* генерируются на одной стороне и передаются другой), где

*p* является [случайным простым числом](https://ru.wikipedia.org/wiki/Случайное_простое_число)

*(p-1)/2* также должно быть [случайным простым числом](https://ru.wikipedia.org/wiki/Случайное_простое_число) (для повышения безопасности)[[5]](https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана#cite_note-5)

*g* является [первообразным корнем](https://ru.wikipedia.org/wiki/Первообразный_корень_(теория_чисел)) [по модулю](https://ru.wikipedia.org/wiki/Сравнение_по_модулю) *p (также является простым числом)*

1. вычисляет *открытый ключ* *A*, используя преобразование над *закрытым ключом*

*A = ga*mod*p*

1. обменивается *открытыми ключами* с удалённой стороной
2. вычисляет *общий секретный ключ* *K*, используя открытый ключ удаленной стороны *B* и свой закрытый ключ *a*

*K = Ba*mod*p*

*К* получается равным с обеих сторон, потому что:

*Ba*mod*p = (gb*mod*p)a*mod*p =****gab*mod*p****= (ga*mod*p)b*mod*p = Ab*mod*p*

В практических реализациях для *a* и *b* используются числа порядка 10100 и *p* порядка 10300. Число *g* не обязано быть большим и обычно имеет значение в пределах первого десятка.

## https://upload.wikimedia.org/wikipedia/commons/thumb/1/13/Diffie-Hellman-Schl%C3%BCsselaustausch.svg/1024px-Diffie-Hellman-Schl%C3%BCsselaustausch.svg.png

## Пример

Ева — криптоаналитик. Она читает пересылку Боба и Алисы, но не изменяет содержимого их сообщений.

* s = секретный ключ. s = 2
* g = первообразный корень по модулю р. g = 5
* p = открытое простое число. p = 23
* a = секретный ключ Алисы. a = 6
* A = открытый ключ Алисы. A = ga mod p = 8
* b = секретный ключ Боба. b = 15
* B = открытый ключ Боба. B = gb mod p = 19

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Alice** | | | Знает | Не знает | | p = 23 | **b** = **?** | | g = 5 |  | | **a** = **6** |  | | A = 5**6** mod 23 = 8 |  | | B = 5**b** mod 23 = 19 |  | | **s** = 19**6** mod 23 = **2** |  | | **s** = 8**b** mod 23 = **2** |  | | **s** = 19**6** mod 23 = 8**b** mod 23 |  | | **s** = **2** |  | | |  |  | | --- | --- | | **Bob** | | | Знает | Не знает | | p = 23 | **a** = **?** | | g = 5 |  | | **b** = **15** |  | | B = 5**15** mod 23 = 19 |  | | A = 5**a** mod 23 = 8 |  | | **s** = 8**15** mod 23 = **2** |  | | **s** = 19**a** mod 23 = **2** |  | | **s** = 8**15** mod 23 = 19**a** mod 23 |  | | **s** = **2** |  | | |  |  | | --- | --- | | **Eve** | | | Знает | Не знает | | p = 23 | **a** = **?** | | g = 5 | **b** = **?** | |  | **s** = **?** | | A = 5**a** mod 23 = 8 |  | | B = 5**b** mod 23 = 19 |  | | **s** = 19**a** mod 23 |  | | **s** = 8**b** mod 23 |  | | **s** = 19**a** mod 23 = 8**b** mod 23 |  | |

## Шифрование с открытым ключом:

Алгоритм Диффи — Хеллмана также может быть использован при шифровании с открытым ключом. В этом случае общая схема остаётся аналогичной приведённой выше, но с небольшими отличиями. Алиса не передаёт значения p, g и A Бобу напрямую, а публикует их заранее в качестве своего открытого ключа. Боб выполняет свою часть вычислений, после чего шифрует сообщение симметричным алгоритмом, используя K в качестве ключа, и передает шифротекст Алисе вместе со значением B.

На практике алгоритм Диффи — Хеллмана таким образом не используется. В данной области доминирующим алгоритмом с открытым ключом является RSA. Это обусловлено больше коммерческими соображениями, так как именно компанией RSA Data Security был создан центр сертификации. К тому же алгоритм Диффи — Хеллмана не может быть использован при подписании сертификатов.

## Получение ключа без передачи ключа:

Если имеется сообщество пользователей, каждый из пользователей может опубликовать открытый ключ в общей базе данных. Если Алиса хочет установить связь с Бобом, ей надо получить открытый ключ Боба и сгенерировать их общий секретный ключ. Алиса может зашифровать сообщение открытым ключом и послать его Бобу. Боб извлечет открытый ключ Алисы и найдет общий секретный ключ.

Каждая пара пользователей может использовать свой уникальный секретный ключ, не требуя обмена данными между пользователями. При этом все открытые ключи должны пройти проверку подлинности для того, чтобы исключить «человека посередине».

# Выводы

# В результате выполнения лабораторной работы была получена реализация алгоритма Деффи-Хеллмана, а также алгоритм генерации ключей для данного алгоритма. Данный алгоритм относится к семейству симметричных алгоритмов.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

class ECC(object):

def \_\_init\_\_(self, a, b, p, base\_point):

self.curve = (a,b,p)

self.base\_point = base\_point

self.double\_base\_point = self.double\_point(base\_point)

def xgcd(self, b, n):

x0, x1, y0, y1 = 1, 0, 0, 1

while n != 0:

q, b, n = b // n, n, b % n

x0, x1 = x1, x0 - q \* x1

y0, y1 = y1, y0 - q \* y1

return b, x0, y0

def inverse(self, b):

g, x, \_ = self.xgcd(b, self.curve[2])

if g == 1:

return x % self.curve[2]

def add\_points(self, p, q):

delta = 0

if p == None or q == None:

return p if q == None else q

elif p[0] == q[0] and p[1] == q[1]:

delta = (3 \* p[0]\*\*2 + self.curve[0]) \* \

self.inverse(2 \* p[1]) % self.curve[2]

else:

delta = (p[1] - q[1]) \* self.inverse((p[0] - q[0])) % self.curve[2]

x = (delta \* delta - p[0] - q[0]) % self.curve[2]

y = (delta \* (p[0] - x) - p[1]) % self.curve[2]

return (x,y)

def double\_point(self, p, k = 1):

Q = p

for i in range(0,k):

Q = self.add\_points(Q,Q)

return Q

def base\_point\_mult(self, k):

Q = None

for i in [1 if digit == '1' else 0 for digit in bin(k)[2:]]:

Q = self.double\_point(Q)

if i == 1:

Q = self.add\_points(Q, self.base\_point)

return Q

class DiffieHellman(object):

def \_\_init\_\_(self, elliptic\_curve, point\_g):

self.elliptic\_curve = elliptic\_curve

self.point\_g = point\_g

def generate\_public\_key(self, private\_key):

if private\_key < 0:

raise ValueError

return self.elliptic\_curve.double\_point(self.point\_g, private\_key)

def secret\_key(self, private\_key, public\_key):

return self.elliptic\_curve.double\_point(public\_key, private\_key)[0]