Протокол Ди́ффи — Хе́ллмана (англ. Diffie–Hellman key exchange protocol, DH) — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования. Данный алгоритм позволил дать ответ на главный вопрос: «Как при обмене зашифрованными посланиями уйти от необходимости передачи секретного кода расшифровки, который, как правило, не меньше самого послания?» Открытое распространение ключей Диффи — Хеллмана позволяет паре пользователей системы выработать общий секретный ключ, не обмениваясь секретными данными.

Предположим, существует два абонента: Алиса и Боб. Алиса и Боб могут обмениваться сообщениями по каналу, который прослушивается. Любое их сообщение перехватывается Евой. Секрет состоит в выборе общего секретного ключа, и чтобы Ева не получила его копию.

Обоим абонентам известны некоторые два числа g и p, которые не являются секретными и могут быть известны также Еве (g = первообразный корень по модулю р. g = 3, p = открытое простое число. p = 17).

a\_public=3

b\_public=17

message="This is a very secret message!!!"

Alice = DH\_Endpoint(a\_public, b\_public, a\_private)

Bob = DH\_Endpoint(a\_public, b\_public, b\_private)

Алиса выбирает случайное приватное число, a = секретный ключ Алисы. a = 54.

a\_private=54

Затем Алиса вычисляет остаток от деления:



def generate\_partial\_key(self):

partial\_key = self.public\_key1\*\*self.private\_key

partial\_key = partial\_key%self.public\_key2

return partial\_key

и пересылает этот результат публично Бобу:

a\_partial=Alice.generate\_partial\_key()

print(a\_partial) #15

Боб выбирает свое приватное число, b = секретный ключ Боба. b = 24.

b\_private=24

И вычисляет остаток от деления:



и передаёт Алисе:

b\_partial=Bob.generate\_partial\_key()

print(b\_partial)

Предполагается, что Ева может получить оба этих значения, но не модифицировать их (то есть, у нее нет возможности вмешаться в процесс передачи).

Теперь, когда Алиса и Боб успешно обменялись частичными ключами, снова применим оператор «возведения в степень» и «деления по модулю» (power и modulo) к частичному ключу другого человека, используя собственные закрытые ключи следующим образом:

def generate\_full\_key(self, partial\_key\_r):

full\_key = partial\_key\_r\*\*self.private\_key

full\_key = full\_key%self.public\_key2

self.full\_key = full\_key

return full\_key

Далее Алиса берет публичный результат Боба и возводит его в степень своего приватного числа:

И получает общий секретный ключ: 1.

a\_full=Alice.generate\_full\_key(b\_partial)

print(a\_full) #1

Боб берет публичный результат Алисы и возводит его в степень своего приватного числа:

И получает тот же самый общий секретный ключ: 1.

b\_full=Bob.generate\_full\_key(a\_partial)

print(b\_full) #1

Оба они произвели одинаковые вычисления, только степени были в разном порядке.

Теперь, когда есть общий ключ шифрования, пришло время начать обмен сообщениями. Простой алгоритм шифрования, чтобы зашифровать сообщение Боба:

def encrypt\_message(self, message):

encrypted\_message = ""

key = self.full\_key

for c in message:

encrypted\_message += chr(ord(c)+key)

return encrypted\_message

Каждый символ кодируется в целое число, добавляется значение ключа, а затем целое число преобразуется обратно в символ. Этот процесс повторяется для каждого символа в сообщении.

b\_encrypted=Bob.encrypt\_message(message)

print(b\_encrypted) #'\x9f³´¾k´¾k¬kÁ°½Äk¾°®½°¿k¸°¾¾¬²°lll'

Теперь пришло время расшифровать сообщение:

def decrypt\_message(self, encrypted\_message):

decrypted\_message = ""

key = self.full\_key

for c in encrypted\_message:

decrypted\_message += chr(ord(c)-key)

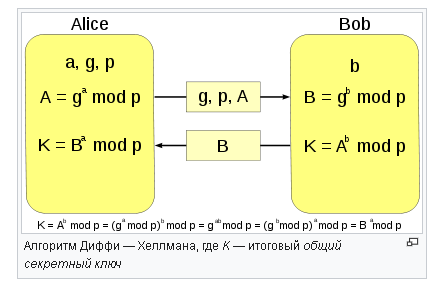
return decrypted\_message

Эта функция делает обратное. В то время как шаг шифрования добавляет значение ключа, шаг расшифровки вычитает то же значение ключа.

message = Alice.decrypt\_message(b\_encrypted)

print(message) #'This is a very secret message!!!'

Общая схема работы алгоритма:



Базовый класс ниже конечной точки, использующей шифрование ДХ:

class DH\_Endpoint(object):

def \_\_init\_\_(self, public\_key1, public\_key2, private\_key):

self.public\_key1 = public\_key1

self.public\_key2 = public\_key2

self.private\_key = private\_key

self.full\_key = None

def generate\_partial\_key(self):

partial\_key = self.public\_key1\*\*self.private\_key

partial\_key = partial\_key%self.public\_key2

return partial\_key

def generate\_full\_key(self, partial\_key\_r):

full\_key = partial\_key\_r\*\*self.private\_key

full\_key = full\_key%self.public\_key2

self.full\_key = full\_key

return full\_key

def encrypt\_message(self, message):

encrypted\_message = ""

key = self.full\_key

for c in message:

encrypted\_message += chr(ord(c)+key)

return encrypted\_message

def decrypt\_message(self, encrypted\_message):

decrypted\_message = ""

key = self.full\_key

for c in encrypted\_message:

decrypted\_message += chr(ord(c)-key)

return decrypted\_message