

# DIFFIE HELMANN KEY EXCHANGE

## Troca de chaves Diffie Hellmann

A troca de chaves de Diffie-Hellman é um método de criptografia específico para troca de chaves desenvolvido por Whitfield Diffie e Martin Hellman e publicado em 1976. Utiliza a dificuldade de calcular de logaritmos discretos em um campo infinito.

O esquema de acordo de chaves Diffie-Hellman não está limitado à interação entre apenas dois participantes. Qualquer número de usuários pode participar no acordo, executando as interações do protocolo e trocando os dados intermediários entre si.

## Algoritmo

### Alice e Bob

Alice e Bob acordam em um número primo  $p$  e uma raiz primitiva modulo  $p$  maior que 2, ambos públicos.

Alice escolhe um número natural aleatório  $a$  e envia  $g^a$  para Bob.

Bob também escolhe um número natural aleatório  $b$  e envia  $g^b$  para Alice.

Alice calcula  $(g^b)^a$ .

Bob calcula  $(g^a)^b$ .

Alice e Bob possuem  $g^{ab}$ , que pode servir como a chave secreta compartilhada.

Os valores de  $(g^b)^a$  e  $(g^a)^b$  são os mesmos porque os grupos são associativos a potência.

Para decifrar uma mensagem  $m$ , enviada como  $mg^{ab}$ , deve primeiro calcular  $(g^{ab})^{-1}$ , da seguinte maneira:

Bob sabe  $|G|$ ,  $b$ , e  $g^a$ . Um resultado da teoria de grupos estabelece que a partir da construção de  $G$ ,  $x^{|G|} = 1$  para todo  $x$  em  $G$ .

Bob então calcula  $(g^a)^{|G|-b} = g^{a(|G|-b)} = g^{a|G|-ab} = g^{a|G|}g^{-ab} = ((g^{|G|})^a)g^{-ab} = (1^a)g^{-ab} = g^{-ab} = (g^{ab})^{-1}$ .

Quando Alice envia a Bob a mensagem encriptada,  $mg^{ab}$ , Bob aplica  $(g^{ab})^{-1}$  e recupera  $mg^{ab}(g^{ab})^{-1} = m(1) = m$ .

### Alice Bob e Carol

Alice, Bob e Carol participam de um acordo do tipo Diffie-Hellman de acordo com o exemplo a seguir, onde todas as operações tomadas com módulo  $p$ :

Os participantes selecionam os parâmetros iniciais do algoritmo  $p$  e  $g$ ; Os participantes geram suas respectivas chaves privadas, que chamaremos  $a$ ,  $b$  e  $c$ . Alice calcula  $g^a$  e envia

o resultado para Bob. Bob calcula  $(g^a)^b = g^{ab}$  e envia o resultado para Carol. Carol calcula  $(g^{ab})^c = g^{abc}$  e utiliza esse valor como sua chave secreta compartilhada. Bob calcula  $g^b$  e envia para Carol. Carol calcula  $(g^b)^c = g^{bc}$  e envia o resultado para Alice. Alice calcula  $(g^{bc})^a = g^{bca} = g^{abc}$  e utiliza o resultado como chave secreta compartilhada. Carol calcula  $g^c$  e envia para Alice. Alice calcula  $(g^c)^a = g^{ca}$  e envia o resultado para Bob. Bob calcula  $(g^{ca})^b = g^{cab} = g^{abc}$  e utiliza o resultado como chave secreta compartilhada.

Um intruso com acesso ao canal onde essas mensagens foram trocadas foi capaz de observar os valores  $g^a, g^b, g^c, g^{ab}, g^{ac}$ , e  $g^{bc}$ , mas não pode usar qualquer combinação desses valores para reproduzir  $g^{abc}$ .

## Exemplos

Alice e Bob entram em acordo para usar um número primo  $p = 23$  e como base  $g = 5$ .

Alice escolhe um inteiro secreto  $a = 6$ , e então envia a Bob  $A = g^a \mod p$ .

$$A = 5^6 \mod 23$$

$$A = 8$$

Bob escolhe um inteiro secreto  $b = 15$ , e então envia a Alice  $B = g^b \mod p$

$$B = 5^{15} \mod 23$$

$$B = 19$$

Alice calcula  $s = B^a \mod p$

$$s = 19^6 \mod 23$$

$$s = 2$$

Bob calcula  $s = A^b \mod p$

$$s = 8^{15} \mod 23$$

$$s = 2$$

Alice e Bob compartilham agora uma chave secreta:  $s = 2$ . Isto é possível porque  $6 \cdot 15$  é o mesmo que  $15 \cdot 6$ . Alguém que tenha descoberto estes dois inteiros privados também será capaz de calcular  $s$  da seguinte maneira:

$$s = 5^{6 \cdot 15} \mod 23$$

$$s = 5^{15 \cdot 6} \mod 23$$

$$s = 5^{90} \mod 23$$

$$s = 2$$

# Código

Listing 1: Diffie Helmann utilizando Miller Rabin em Python.

---

```
1 import random
2 import sys
3 import math
4
5 """
6 Decompose um numero par na forma (2^r) * s
7 """
8 def decomposeBaseTwo(n):
9     exponentOfTwo = 0
10    while n % 2 == 0:
11        n = n/2
12        exponentOfTwo += 1
13
14    return exponentOfTwo, n
15
16 """
17 Verifica as condicoes
18     Se (a^s == 1 (mod n) ou a^2js == -1 (mod n)
19     para um j | 0 <= j <= r-1
20 """
21 def fillPrimeConditions(candidateNumber, p, exponent, remainder):
22     candidateNumber = pow(candidateNumber, remainder, p)
23
24     if candidateNumber == 1 or candidateNumber == p - 1:
25         return False
26
27     for _ in range(exponent):
28         candidateNumber = pow(candidateNumber, 2, p)
29
30         if candidateNumber == p - 1:
31             return False
32
33     return True
34
35 """
36     O numero randomico a na faixa que inicia em 2 pois, o teste 1^s = 1(mod n)
37     Seria uma tentativa inutil
38 """
39 def probablyPrime(p, accuracy=100):
40     if p == 2 or p == 3: return True
41     if p < 2: return False
42
43     numTries = 0
44     exponent, remainder = decomposeBaseTwo(p - 1)
45
46     for _ in range(accuracy):
47         candidateNumber = random.randint(2, p - 2)
48         if fillPrimeConditions(candidateNumber, p, exponent, remainder):
49             return False
50
51     return True
52
53 def generateRandomPrime(bits, precision):
54     random_number = random.getrandbits(bits)
```

```

55     while (probablyPrime(random_number, precision) == False):
56         random_number = random.getrandbits(bits)
57     return random_number
58
59
60 def bigRange(start, stop, step):
61     i = start
62     while i < stop:
63         yield i
64         i += step
65
66 def primeFactors(n):
67     f, fs = 3, set()
68     while n % 2 == 0:
69         fs.add(2)
70         n /= 2
71     while f * f <= n:
72         while n % f == 0:
73             fs.add(f)
74             n /= f
75         f += 2
76     if n > 1: fs.add(n)
77     return fs
78
79 def primitiveRoots(number):
80     if number:
81         if number == 2:
82             yield 1
83         if number == 3:
84             yield 2
85         phi = number - 1
86         factors = primeFactors(phi)
87         for m in bigRange(2, phi, 1):
88             is_root = True
89             for p in factors:
90                 if pow(m, int(phi / p), number) == 1:
91                     is_root = False
92             if (is_root):
93                 yield m
94
95 def diffie_hellmann(bits, precision):
96     prime = generateRandomPrime(bits, precision)
97     for root in primitiveRoots(prime):
98         if root == 2:
99             break
100     XA = random.getrandbits(bits - 1)
101     XB = random.getrandbits(bits - 1)
102     YA = pow(root, XA, prime)
103     YB = pow(root, XB, prime)
104
105     K1 = pow(YB, XA, prime)
106     K2 = pow(YA, XB, prime)
107     return
108
109
110 def main():
111     bits = int(sys.argv[1])
112     precision = int(sys.argv[2])
113     diffie_hellmann(bits, precision)

```

```
114
115 if __name__ == '__main__':
116     main()
```

---

## Execução

Para executar o script basta abrir um terminal e utilizar um shell tipo o bash, o script é interativo.

Listing 2: Executando o script.

---

```
1 python diffie_hellmann.py
```

---