Diffie Helmann Key Exchange

Rafael de Lucena Valle, Universidade Federal de Santa Catarina 10/06/2014

Troca de chaves Diffie Helmann

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-Helman não está limitado à interação entre apenas dois participantes. Qualquer número de usuários pode participar no acordo, executanto 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 quanto 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 exeomplo 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 resutlado 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*15 é o mesmo que 15*6. Alguém que tenha descoberto estes dois inteiros privados também será capaz de calcular s da seguinte maneira:
$s = 5^{6*15} \mod 23$
$s = 5^{15*6} \mod 23$
$s = 5^{90} \mod 23$
$s = 2$

Código

Listing 1: Diffie Helmann utilizando Miller Rabin em Python.

```
import random
import sys
import math

"""
Decompoe um numero par na forma (2^r) * s
"""
def decomposeBaseTwo(n):
    exponentOfTwo = 0
    while n % 2 == 0:
      n = n/2
      exponentOfTwo += 1

    return exponentOfTwo, n

"""
Verifica as condicoes
    Se (a^s === 1 (mod n) ou a^2js === -1 (mod n)
    para um j | 0 <= j <= r-1
"""
def fillPrimeConditions(candidateNumber, p, exponent, remainder):
   candidateNumber = pow(candidateNumber, remainder, p)

   if candidateNumber == 1 or candidateNumber == p - 1:
      return False

   for _ in range(exponent):
      candidateNumber = pow(candidateNumber, 2, p)

      if candidateNumber == p - 1:
         return False

   return True

"""
  O numero randomico a na faixa que inicia em 2 pois, o teste 1^s = 1(mod n)
  Seria uma tentavia inutil
"""
def probablyPrime(p, accuracy=100):
   if p == 2 or p == 3: return True
   if p < 2: return False

   numTries = 0
   exponent, remainder = decomposeBaseTwo(p - 1)

   for _ in range(accuracy):
      candidateNumber = random.randint(2, p - 2)
      if fillPrimeConditions(candidateNumber, p, exponent, remainder):
         return False

   return True

def generateRandomPrime(bits, precision):
    random_number = random.getrandbits(bits)
```

```python
    while (probablyPrime(random_number, precision) == False):
        random_number = random.getrandbits(bits)
    return random_number


def bigRange(start, stop, step):
    i = start
    while i < stop:
        yield i
        i += step

def primeFactors(n):
    f, fs = 3, set()
    while n % 2 == 0:
        fs.add(2)
        n /= 2
    while f * f <= n:
        while n % f == 0:
            fs.add(f)
            n /= f
        f += 2
    if n > 1: fs.add(n)
    return fs

def primitiveRoots(number):
    if number:
        if number == 2:
            yield 1
        if number == 3:
            yield 2
        phi = number - 1
        factors = primeFactors(phi)
        for m in bigRange(2, phi, 1):
            is_root = True
            for p in factors:
                if pow(m, int(phi / p), number) == 1:
                    is_root = False
            if (is_root):
                yield m

def diffie_helmann(bits, precision):
    prime = generateRandomPrime(bits, precision)
    for root in primitiveRoots(prime):
        if root == 2:
            break
        XA = random.getrandbits(bits - 1)
        XB = random.getrandbits(bits - 1)
        YA = pow(root, XA, prime)
        YB = pow(root, XB, prime)

        K1 = pow(YB, XA, prime)
        K2 = pow(YA, XB, prime)
        return


def main():
    bits = int(sys.argv[1])
    precision = int (sys.argv[2])
    diffie_helmann(bits, precision)
```

```python

if __name__ == '__main__':
    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.

```
python diffie_helmann.py

#    Output Example:
#
#    Give the size of prime random number in bits: 40
#    Which precision to test primality? 1000
#    --------------------------------------------------------
#    Prime P: 898567660219
#    Primitive Root Mod P: 3
#    Private Keys XA: 341287795542 and XB: 519787381445
#    Public Keys YA: 877816592258 and YB: 663134792560
#    K1: 121271137292 K2: 121271137292
#    --------------------------------------------------------
```