Intercambio de claves de Diffie y Hellman

Adrián Racero Serrano
Juan Manuel Cardeñosa Borrego

Índice

1 Introducción

El algoritmo Diffie-Hellman debe su nombre a sus creadores Whitfield Diffie y Martin Hellman. Creado en 1976, es uno de los protocolos de intercambio de claves más antiguos que todavía se siguen usando en la actualidad. Sus creadores fueron galardonados con el premio A.M. Turing 2015 por este trabajo, con el que revolucionaron por completo la seguridad informática.

Este algoritmo permite a dos usuarios cualesquiera intercambiar, de forma confidencial, una clave secreta K (o de sesión) para posteriormente cifrar de forma simétrica los mensajes entre ellos dos.

2 Algoritmo

El funcionamiento de este algoritmo es más sencillo de lo que parece y se usa frecuentemente en protocolos y aplicaciones de encriptado de datos, como SSL (Secure Sockets Layer), SSH (Secure Shell) o VPN (Virtual Private Network). Este algoritmo permite que dos entidades (A y B) puedan generar una clave K_{AB} de forma simultánea, y sin enviarla por el canal de comunicaciones.

1) Para ello, A y B necesitan establecer y compartir valores comunes, como un valor q primo y una raíz primitiva α de q.

Para todo primo q existe un elemento $\alpha \in (\mathbb{Z}/q\mathbb{Z})^\times$ con $\text{ord}(\alpha) = q - 1 = \phi(q) = \#(\mathbb{Z}/q\mathbb{Z})^\times$.

$(\mathbb{Z}/q\mathbb{Z})^\times$ es el grupo multiplicativo de las clases de equivalencia módulo q, es decir, que no tienen ningún factor primo en común con q:

$$(\mathbb{Z}/q\mathbb{Z})^\times = \{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}.$$

Este α se llama una **raíz primitiva** módulo q. Ejemplo:

$$q = 3,\ \alpha = 2;\ 2^1\ mod\ 3 = 1,\ 2^2\ mod\ 3 = 2$$

2) Tanto A como B generan sus claves privadas ($X_{A/B}$) y públicas ($Y_{A/B}$) teniendo en cuenta que: $Y_A \equiv \alpha^{XA} \pmod q$, donde $0 \leq X_A \leq (q-1)$, X_A es el logaritmo discreto de Y_A, y se representa $dlog_{\alpha,q}\ (Y_A)$. Por tanto, la efectividad del algoritmo depende de la dificultad de computar logaritmos discretos.

3) Tanto A como B comparten sus respectivas claves públicas (Y_A/Y_B).

4) Tanto A como B generan la clave de sesión teniendo en cuenta:

$$K_{AB} = (Y_B)^{X_A} \mod q \rightarrow K_{AB} = (\alpha^{X_B})^{X_A} \mod q$$
$$K_{AB} = (Y_A)^{X_B} \mod q \rightarrow K_{AB} = (\alpha^{X_A})^{X_B} \mod q$$

Por tanto:

$$K_{AB} = \alpha^{X_B X_A} \mod q = \alpha^{X_A X_B}$$

3 Código

La clase **DiffieHellman**:

```
1  from funciones_auxiliares import *
2
3  class DiffieHellman :
4
5      def __init__(self, q, alfa):
6
7          if not(es_primo(q)):
8              print("ERROR: q no es primo")
9              exit(-1)
10
11         if not(es_raiz_primitiva(alfa, q)):
12             print("ERROR: alfa no es raiz primitiva de
                   q")
13             exit(-1)
14
15         self.__q__ = q
16         self.__alfa__ = alfa
17
18         self.__generarClaves__()
19
20     def __generarClaves__(self):
21         # X < q
22         self.__clavePrivada__ = numero_aleatorio(self.
               __q__)
23         # Y = alfa ^X mod q
24         self.__clavePublica__ = ( self.__alfa__ **
               self.__clavePrivada__ ) % self.__q__
25
26     def compartir_clave_publica(self):
27         return self.__clavePublica__
28
29     def generar_clave_secreta(self, clavePublicaB):
30         # K = Yb ^Xa mod q
31         self.__claveSecreta__ = ( clavePublicaB **
               self.__clavePrivada__ ) % self.__q__
32
33     # Este metodo solo se utiliza para comprobar
34     # que ambas claves secretas son iguales.
```

```
35      # Las claves secretas no se deben compartir.
36      def compartir_clave_secreta(self):
37          return self.__claveSecreta__
38
39      # Cifrar un mensaje usando XOR y la clave secreta
40      def cifrar_xor(self, mensaje):
41          cifrado = [char ^ self.__claveSecreta__ for
                char in mensaje.encode()]
42          return bytes(cifrado)
43
44      # Descifrar un mensaje usando XOR y la clave
            secreta
45      def descifrar_xor(self, cifrado):
46          mensaje = ''.join([chr(char ^ self.
                __claveSecreta__) for char in cifrado])
47          return mensaje
```

Ejecucion del algoritmo:

```
1  from algoritmo import *
2
3  print("------ INICIALIZAR VALORES ------")
4  print("q = 153")
5  print("alfa = 3")
6  print()
7  DH1 = DiffieHellman(353,3)
8  DH2 = DiffieHellman(353,3)
9
10 print("------ GENERAR CLAVE SECRETA ------")
11 DH1.generar_clave_secreta(DH2.compartir_clave_publica
       ())
12 print("Clave de DH1: " + str(DH1.
       compartir_clave_secreta()))
13 DH2.generar_clave_secreta(DH1.compartir_clave_publica
       ())
14 print("Clave de DH2: " + str(DH2.
       compartir_clave_secreta()))
15 print()
16
17 print("------- CIFRADO/DESCIFRADO -------")
18 mensaje = "Diffie Hellman"
19 cifrado = DH1.cifrar_xor(mensaje)
20 print("DH1 cifrando...")
21 print(mensaje + " -> " + str(cifrado))
22 mensaje_descifrado = DH2.descifrar_xor(cifrado)
23 print("DH2 descifrando...")
```

```
24 print(str(cifrado) + " -> " + mensaje_descifrado)
```

Salida de la ejecución:

```
------- INICIALIZAR VALORES -------
q = 153
alfa = 3

------- GENERAR CLAVE SECRETA -------
Clave de DH1: 231
Clave de DH2: 231

------- CIFRADO/DESCIFRADO -------
DH1 cifrando...
Diffie Hellman -> b'\xa3\x8e\x81\x81\x8e\x82\xc7\xaf
\x82\x8b\x8b\x8a\x86\x89'
DH2 descifrando...
b'\xa3\x8e\x81\x81\x8e\x82\xc7\xaf
\x82\x8b\x8b\x8a\x86\x89' -> Diffie Hellman
```

4 Conclusión

a