Cifrado ElGamal

Canek García (kaan.ek@ciencias.unam.mx)

Agenda

- Breve historia
- Cifrado ElGamal
- Implementación

Breve historia...

- Algoritmo de criptografía asimétrico basado en la idea de **Diffie-Hellman**.
- El algoritmo se utiliza para generar firmas digitales y para cifrar o descifrar.
- Fue descrito por Toher Elgamal en 1984 y se usa en software **GNU Privacy Guard**, versiones de **PGP** y otros sistemas criptográficos.
- El algoritmo no está bajo ninguna patente, lo que lo hace de uso **libre**.
- Su seguridad se basa en la intratabilidad computacional del problema del logaritmo discreto (**DLP**), que consiste en:

Dado un grupo finito cíclico ***G***, un generador ***g*** de ***G*** y un elemento ***h*** de ***G***, encontrar el entero ***x*** tal que $h = g^x$, es decir, el logaritmo discreto de ***h*** en base ***g***.

Cifrado ElGamal

1. Generación de la clave

1. Escoger un **número primo p** cualquiera, tal que el logaritmo discreto no es soluble en un tiempo asumible en $\mathbb{Z}^*_p$ (grupo multiplicativo módulo un primo p). Esto último se traduce en que **(p - 1)** tenga un factor primo grande (lo que hace que el problema de logaritmo discreto sea difícil).
2. Elegir **dos número** aleatorios **g** (el generador del grupo cíclico $\mathbb{Z}^*_p$) y **a** (que actuará como clave privada) tal que **a** $\in$ { 0 ... p-1 }
3. Calcular el valor de $\mathbf{K = g^a \pmod p}$
   La **clave pública** será: **(g, p, K)**
   La **clave privada** será: $\mathbf{\log_g K \pmod p = a}$

2. Cifrado

1. Convertir texto en un entero **m**, entre 1 y p-1 ($m \in \mathbb{Z}_p$)
2. Escoger arbitrariamente un número **b** $\in \{ 2 \dots p-1 \}$
3. Calcular:

$$y_1 = g^b \pmod p$$
$$y_2 = K^b m \pmod p$$

4. El mensaje cifrado corresponde a:

$$C_b(m,b) = (y_1, y_2)$$

3. Descifrado

1. Para descifrar, se realiza el siguiente cálculo:

$$y_1^{-a} y_2 \pmod{p}$$

Implementación

Generación de claves

```java
public static List<List<BigInteger>> KeyGen(int n) {
    // 1. take a random prime 'p' with getPrime() function. p = 2 * p' + 1 with prime(p') = true
    BigInteger p = getPrime(n, 40, new Random());
    // 2. take a random element in [Z/Z[p]]* (p' order)
    BigInteger g = randNum(p, new Random());
    BigInteger pPrime = p.subtract(BigInteger.ONE).divide(ElGamal.TWO);

    while (!g.modPow(pPrime, p).equals(BigInteger.ONE)) {
        if (g.modPow(pPrime.multiply(ElGamal.TWO), p).equals(BigInteger.ONE))
            g = g.modPow(TWO, p);
        else
            g = randNum(p, new Random());
    }

    // 3. take a random in [0, p' - 1]
    BigInteger a = randNum(pPrime.subtract(BigInteger.ONE), new Random());
    // 4. calculate K = g^a (mod p)
    BigInteger K = g.modPow(a, p);
    // secret key is (p, a) and public key is (p, g, K)
    List<BigInteger> sk = new ArrayList<>(Arrays.asList(p, a));
    List<BigInteger> pk = new ArrayList<>(Arrays.asList(p, g, K));
    // [0] = pk, [1] = sk
    return new ArrayList<>(Arrays.asList(pk, sk));
}
```

Cifrado

```
public static List<BigInteger> Encrypt(BigInteger p, BigInteger g, BigInteger K, BigInteger message) {
    BigInteger pPrime = p.subtract(BigInteger.ONE).divide(ElGamal.TWO);
    BigInteger b = randNum(pPrime, new Random());
    // encrypt couple (g^b, m * K^b)
    // i.e. y1 and y2
    return new ArrayList<>(Arrays.asList(g.modPow(b, p), message.multiply(K.modPow(b, p))));
}
```

Descifrado

```
public static BigInteger Decrypt(BigInteger p, BigInteger a, BigInteger y1, BigInteger y2) {
    BigInteger hr = y1.modPow(a, p);
    return y2.multiply(hr.modInverse(p)).mod(p);
}
```