

# Esquema de Firma digital Elgamal

El esquema de firma digital de Elgamal fue publicado en 1985 y se basa en el logaritmo discreto. Sin embargo, este esquema es diferente del protocolo de cifrado.

## Versión de libro de texto

Hay dos fases: generación de llaves, y generación + verificación de firmas.

## Generación de llaves

- 1.- Se escoge un primo  $p$  grande.
- 2.- Se escoge un  $\alpha \in \mathbb{Z}_p^*$  primitivo ó un subgrupo de  $\mathbb{Z}_p^*$ .
- 3.- Se escoge un  $d \in \{2, 3, \dots, p-2\}$  aleatorio.
- 4.- Se calcula  $\beta = \alpha^d \bmod p$ .

Así, la llave pública es  $k_{\text{pub}} = (p, \alpha, \beta)$ , mientras que la llave privada es  $k_{\text{priv}} = d$ .

## Generación de firma

1. Se escoge  $k_E \in \{2, \dots, p-2\}$  aleatorio tq  $\text{gcd}(k_E, p-1) = 1$ ,  
a  $k_E$  se le llama llave efímera.
2. Se calculan  $r \equiv \alpha^{k_E} \pmod{p}$   
 $s \equiv (x - dr) k_E^{-1} \pmod{p}$

Con esto, la firma digital es  $\text{sig}_{k_{\text{priv}}}(x, k_E) = (r, s)$

## Verificación de firma

1. Se calcula  $t \equiv \beta^r r^s \pmod{p}$
2. Se valida la firma si y sólo si  $t \equiv \alpha^x \pmod{p}$ .

Ejemplo:

llave pública

$$\xleftarrow{(p, \alpha, \beta) = (29, 2, 7)}$$

### Bob

1. choose  $p = 29$
2. choose  $\alpha = 2$
3. choose  $d = 12$
4.  $\beta = \alpha^d \equiv 7 \pmod{29}$

compute signature for message  
 $x = 26$ :

choose  $k_E = 5$

(note that  $\text{gcd}(5, 28) = 1$ )

$$r = \alpha^{k_E} \equiv 2^5 \equiv 3 \pmod{29}$$

$$s = (x - dr) k_E^{-1} \\ \equiv (-10) \cdot 17 \equiv 26 \pmod{28}$$

mensaje y firma

$$\xleftarrow{(x, (r, s)) = (26, (3, 26))}$$

verify:

$$t = \beta^r \cdot r^s \equiv 7^3 \cdot 3^{26} \equiv 22 \pmod{29}$$

$$\alpha^x \equiv 2^{26} \equiv 22 \pmod{29}$$

$$t \equiv \alpha^x \pmod{29} \implies \text{valid signature}$$

◇

## Aspectos computacionales

Como la generación de la llave es idéntica al set-up del cifrado Elgamal, ent. ya conocemos restricciones.

En particular,  $p$  debe tener longitud de bits al menos de 2048.

También,  $d$  debe obtenerse con un TRNG

La  $\beta$  se puede obtener con el método de multiplicación y cuadrado.

La firma  $(x, (r, s))$  tiene que  $l(r) \sim l(p) \sim l(s)$

$$\Rightarrow l((x, (r, s))) \sim 3l(x)$$

Para calcular  $r$  se usa el método de multiplicación y cuadrado.

Para calcular  $s$  lo "difícil" es calcular  $k_t^{-1}$ . Esto se puede hacer ya sea con el algoritmo de Euclides extendido o con una tabla precalculada. De hecho, la pareja  $(k_t, r)$  puede precalcularse y guardarse para su uso (incluyendo el cálculo de  $s$ )

La verificación se realiza con el método de multiplicación y cuadrado.

## Seguridad

El primer punto concerniente a la seguridad es análogo al del esquema RSA de firmas: asegurarse que "Alice" tiene la llave pública correcta.

Fuera de esto, se tienen 3 ataques principales

- Cálculo de logaritmo discreto  $l(p) \geq 2048$ ;  $p$  primo
- Reutilización de llave efímera

Si se reutiliza  $k_E$  para dos firmas  $(x_1, (r_1, s_1))$  y  $(x_2, (r_2, s_2))$ , es "obvio" pues  $r_1 \equiv \alpha^{k_E} \equiv r_2 \pmod{p}$

$$\Rightarrow s_1 - s_2 \equiv (x_1 - x_2) k_E^{-1} \pmod{p-1}$$

$$\Rightarrow k_E \equiv \frac{x_1 - x_2}{s_1 - s_2} \pmod{p-1}$$

$\Rightarrow$  el atacante puede calcular (eventualmente)  $k_E$

$$\Rightarrow d = \frac{x_1 - s_1 k_E}{r_1} \pmod{p-1}$$

$\Rightarrow$  el atacante puede calcular (eventualmente)  $d$

- Ataque de imitación existencial

...

### Existential Forgery Attack Against Elgamal Digital Signature

Alice

Oscar

Bob

$$k_{pr} = d$$

$$k_{pub} = (p, \alpha, \beta)$$

$\xleftarrow{(p, \alpha, \beta)}$

$\xleftarrow{(p, \alpha, \beta)}$

1. select integers  $i, j$   
where  $\gcd(j, p-1) = 1$
2. compute signature:  
 $r \equiv \alpha^i \beta^j \pmod{p}$   
 $s \equiv -r j^{-1} \pmod{p-1}$
3. compute message:  
 $x \equiv s i \pmod{p-1}$

$\xleftarrow{(x, (r, s))}$

verification:

$$t \equiv \beta^r \cdot r^s \pmod{p}$$

since  $t \equiv \alpha^x \pmod{p}$ :

valid signature!

El atacante puede crear firmas válidas para mensajes pseudocaleatorios.

Para prevenir esto, se "hashea" el mensaje y se utiliza el mensaje hasheadó para la firma.

## Algoritmo de firma digital

DSA es una variante del algoritmo de firmas Elgamal. Es también conocido como Estándar de firma digital de EUA (US DSS).

El NIST dejó de recomendar el DSA en 2020 y recomienda más el RSA o ECDSA

Las ventajas de DSA sobre Elgamal es que las firmas son cortas en comparación al módulo (448 vs 2048), y que hay ataques de Elgamal que no funcionan en DSA.

### Algoritmo DSA (2048 bits, pero hay de 3072 bits)

## Generación de llaves

- 1.- Se genera un primo  $tq$   $2^{2047} < p < 2^{2048}$
- 2.- Encuentra  $q$  un divisor primo de  $p-1$   $tq$   
 $2^{223} < q < 2^{224}$
- 3.- Tomamos  $\alpha \in \mathbb{Z}_p^*$   $tq$   $\langle \alpha \rangle \cong \mathbb{Z}_q^*$
- 4.- Tomamos  $d$  aleatorio  $tq$   $0 < d < q$
- 5.- Calculamos  $\beta = \alpha^d \pmod{p}$
- 6.- La llave es  $k_{\text{pub}} = (p, q, \alpha, \beta)$ ,  $k_{\text{priv}} = d$

Como tenemos dos grupos cíclicos  $(\mathbb{Z}_p^* \& \mathbb{Z}_q^*)$ , ent. la firma va a resultar pequeña.

Las combinaciones de primos pueden ser más grandes:

$p$	$q$	firma
1024	160	320
2048	224	448
3072	256	512

## Generación de firmas

1. Se escoge una llave efímera  $t_q$   $0 < k_t < q$ .
2. Se calcula  $r \equiv (\alpha^{k_t} \bmod p) \bmod q$
3. Se calcula  $s \equiv (SHA(x) + dr) k_t^{-1} \bmod q$ , donde SHA es la función hash SHA-2

Lo que necesitamos ahorita es saber que SHA-2 comprime  $x$  y calcula una "huella digital" de al menos 224 bits.

## Verificación de firmas

1. Calcular los valores auxiliares  $w \equiv s^{-1} \bmod q$
2. Calcular los valores auxiliares  $u_1 \equiv w \cdot SHA(x) \bmod q$
3. Calcular los valores auxiliares  $u_2 \equiv wr \bmod q$
4. Calcular  $v \equiv (\alpha^{u_1} \beta^{u_2} \bmod p) \bmod q$
5. Se valida la firma si y sólo si  $v \equiv r \bmod q$

¿Qué significa que la firma no se validó? O bien el mensaje o la firma fueron modificados, o el verificador no tiene la llave pública correcta.

¿Por qué funciona la verificación?

$$s \equiv (\text{SHA}(x) + dr) k_E^{-1} \pmod{q}$$

$$\Rightarrow k_E \equiv s \cdot \text{SHA}(x) + s' dr \pmod{q}$$

$$\Rightarrow k_E \equiv u_1 + du_2 \pmod{q}$$

$$\rightsquigarrow \alpha^{k_E} \equiv \alpha^{u_1 + du_2} \pmod{p}$$

$$\equiv \alpha^{u_1} (\alpha^d)^{u_2} \pmod{p}$$

$$\equiv \alpha^{u_1} \beta^{u_2} \pmod{p}$$

$$\Rightarrow (\underbrace{\alpha^{k_E} \pmod{p}}_r) \equiv (\underbrace{\alpha^{u_1} \beta^{u_2} \pmod{p}}_v) \pmod{q}.$$

Ejemplo:

Bob

1. choose  $p = 59$
2. choose  $q = 29$
3. choose  $\alpha = 3$
4. choose private key  $d = 7$
5.  $\beta = \alpha^d \equiv 4 \pmod{59}$

$$\xleftarrow{(p,q,\alpha,\beta)=(59,29,3,4)}$$

sign:

compute hash of message  $h(x) = 26$

1. choose ephemeral key  $k_E = 10$
2.  $r = (3^{10} \pmod{59}) \equiv 20 \pmod{29}$
3.  $s = (26 + 7 \cdot 20) \cdot 3 \equiv 5 \pmod{29}$

$$\xleftarrow{(x,(r,s))=(x,(20,5))}$$

verify:

1.  $w = 5^{-1} \equiv 6 \pmod{29}$
  2.  $u_1 = 6 \cdot 26 \equiv 11 \pmod{29}$
  3.  $u_2 = 6 \cdot 20 \equiv 4 \pmod{29}$
  4.  $v = (3^{11} \cdot 4^4 \pmod{59}) \pmod{29} = 20$
  5.  $v \equiv r \pmod{29} \implies \text{valid signature}$
- 

## Aspectos computacionales

Para este algoritmo, la parte más pesada en realidad es la generación de llave.

El reto es encontrar un  $p$  primo tq  $\ell(p) = 2048$  & que tenga un divisor primo  $q$  de  $p-1$  tq  $\ell(q) \sim 224$ .

Normalmente se procede empezando por  $q$ , calculando de ahí la  $p$ .

## Prime Generation for DSA

**Output:** two primes  $(p, q)$ , where  $2^{2047} < p < 2^{2048}$  and  $2^{223} < q < 2^{224}$ , such that  $p - 1$  is a multiple of  $q$ .

**Initialization:**  $i = 1$

**Algorithm:**

```
1  find prime  $q$  with  $2^{223} < q < 2^{224}$  using the Miller–Rabin algorithm
2  FOR  $i = 1$  TO 4096
   2.1  generate random integer  $M$  with  $2^{2047} < M < 2^{2048}$ 
   2.2   $M_r \equiv M \pmod{2q}$ 
   2.3   $p - 1 \equiv M - M_r$           (note that  $p - 1$  is a multiple of  $2q$ .)
        IF  $p$  is prime          (use Miller–Rabin primality test)
        2.4    RETURN  $(p, q)$ 
   2.5   $i = i + 1$ 
3  GOTO Step 1
```

En comparación, el resto es "sencillo".

$r = \alpha^{k_E} \rightsquigarrow$  multiplicación y cuadrado  $\sim 336$  requeridas

Se obtiene un resultado de 2048 bits

$\rightsquigarrow$  al hacer  $\mod q$  se tiene un resultado de 224 bits

Para  $s$  se opera solo con 224 bits y lo más "raro" es  $k_E^{-1}$ .

Notemos  $r$  y  $k_E$  se pueden precalcular.

Ya la verificación se hace en 224 bits, y lo más "caro" es la exponentiación.

## Seguridad

La primera forma de atacar es resolver

$$d \equiv \log_{\alpha} \beta \pmod{p}$$

$$\Rightarrow l(p) \geq 2048$$

La segunda forma es usar que  $\langle \alpha \rangle = \mathbb{Z}_q^*$  el cual es un subgrupo chico en comparación. Sin embargo, el ataque clásico para esto no es realmente aplicable. Los ataques clásicos aquí nos dejan una seguridad de  $2^{112}$

**Table 10.2** Bit lengths and security levels for DSA

$p$	$q$	Hash output (min)	Security levels
1024	160	160	80
2048	224	224	112
3072	256	256	128

Cabe mencionar que, al igual que en Elgamal, la reutilización de las llaves efímeras rompe la seguridad del algoritmo.