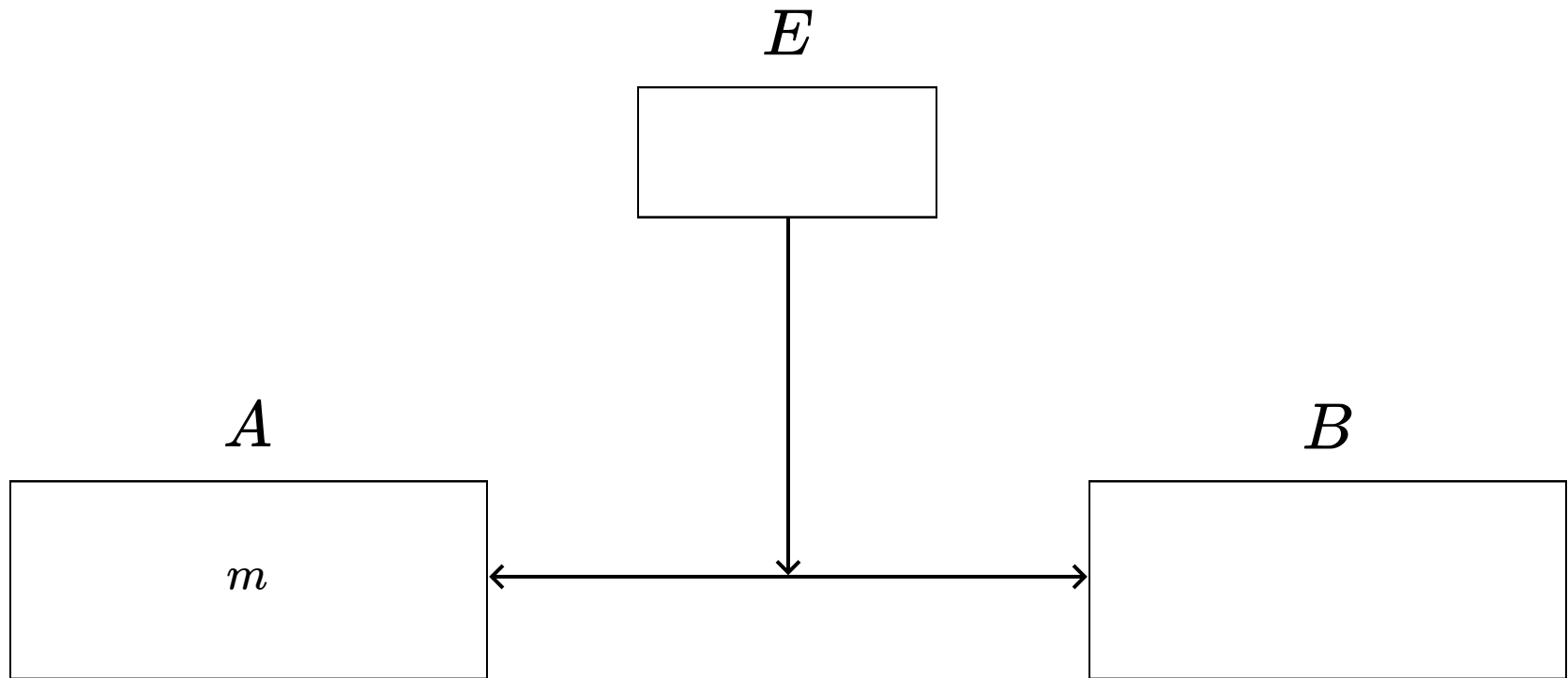


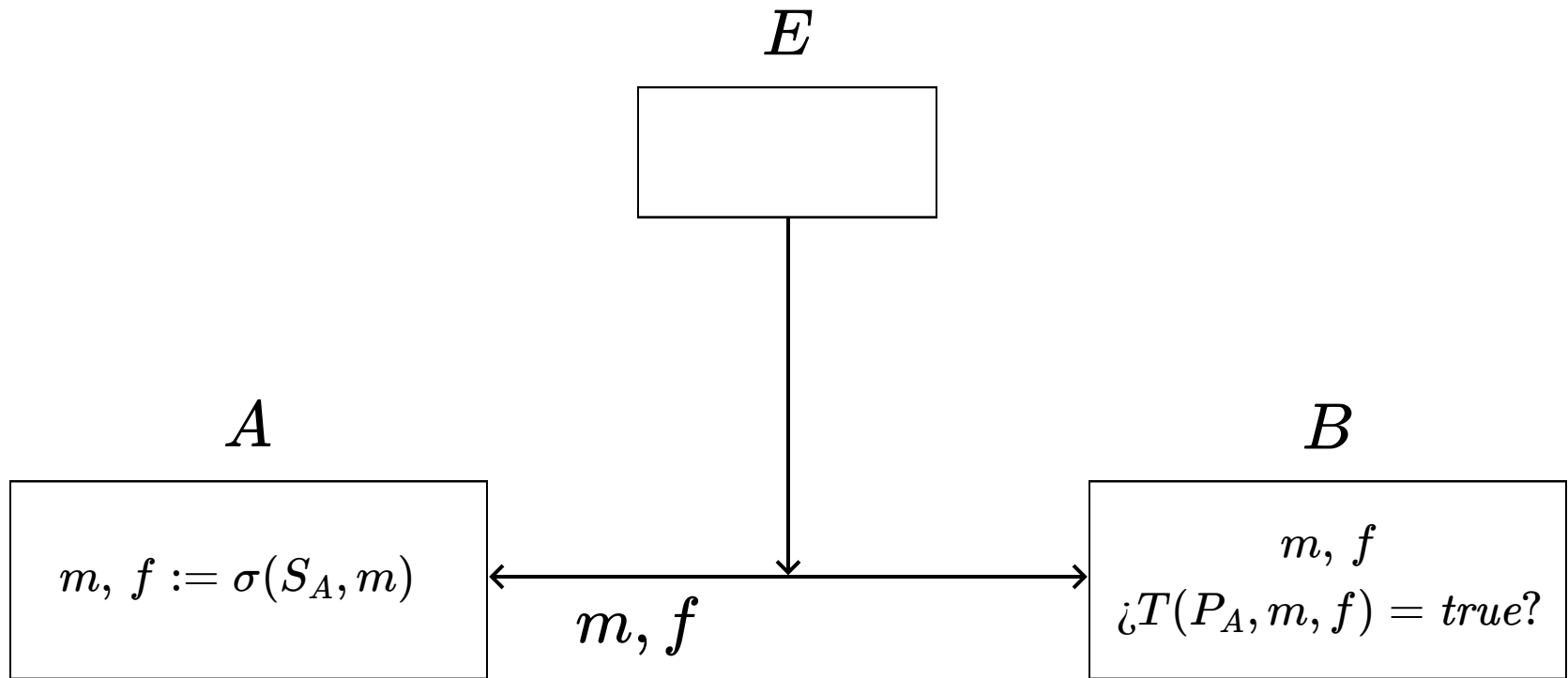
# IIC3253

Firmas digitales

# Firma digital con una clave pública



# Firma digital con una clave pública



# Firma digital con una clave pública

- $A$  está firmando un mensaje  $m$ , para cualquiera que lo necesite
- $\sigma(S_A, m)$  utiliza la clave secreta de  $A$  para generar una firma  $f$  de  $m$ , de manera tal que solo  $A$  puede firmar
- $T(P_A, m, f)$  verifica si  $f$  es una firma válida del mensaje  $m$  por el usuario  $A$
- $T(P_A, m, f)$  utiliza la clave pública de  $A$ , de manera que cualquiera puede verificar si  $f$  es una firma válida

# Firmas digitales con RSA

Suponga que  $P_A = (e, N)$  y  $S_A = (d, N)$  son las claves pública y privada de un usuario  $A$

Para cada  $m \in \{0, \dots, N - 1\}$ , sabemos que

$$Dec(S_A, Enc(P_A, m)) = m$$

# Firmas digitales con RSA

Pero también tenemos que

$$Enc(P_A, Dec(S_A, m)) =$$

# Firmas digitales con RSA

Pero también tenemos que

$$\begin{aligned} \text{Enc}(P_A, \text{Dec}(S_A, m)) &= (m^d \bmod N)^e \bmod N \\ &= (m^d)^e \bmod N \\ &= m^{d \cdot e} \bmod N \\ &= (m^e)^d \bmod N \\ &= (m^e \bmod N)^d \bmod N \\ &= \text{Dec}(S_A, \text{Enc}(P_A, m)) \\ &= m \end{aligned}$$

# Firmas digitales con RSA

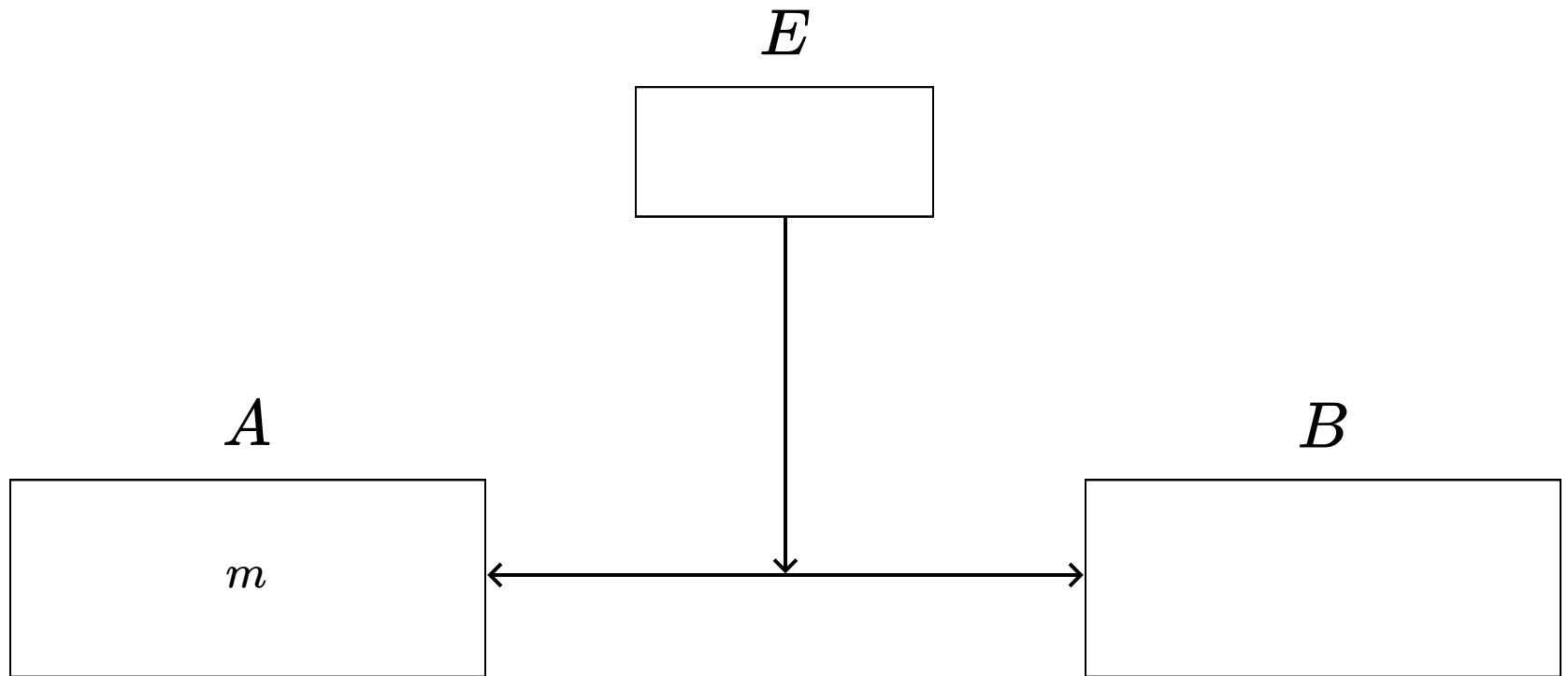
Definimos entonces la firma del mensaje  $m$  por el usuario  $A$  como:

$$f := Dec(S_A, m)$$

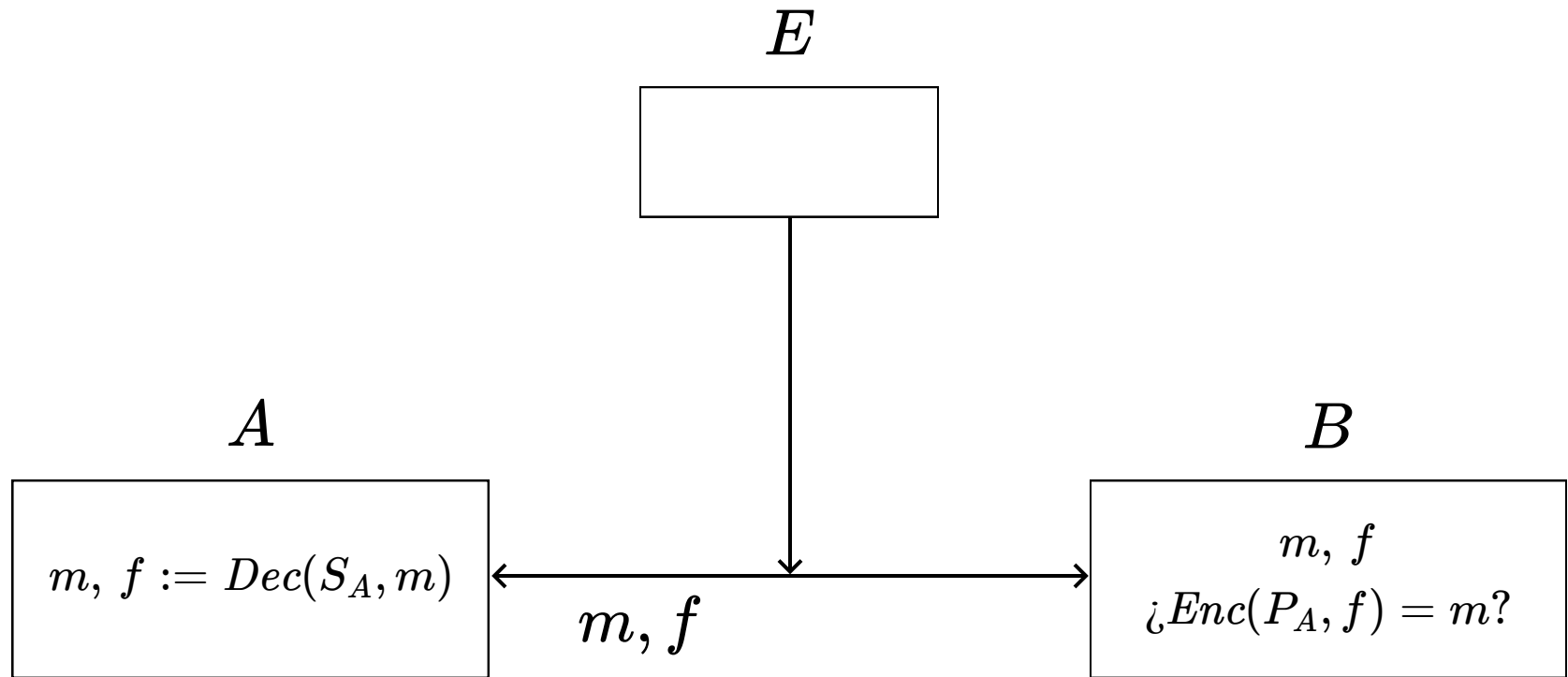
Solo  $A$  puede generar esta firma. Cualquier usuario puede verificar si  $A$  firmó un mensaje usando la clave pública  $P_A$  de  $A$



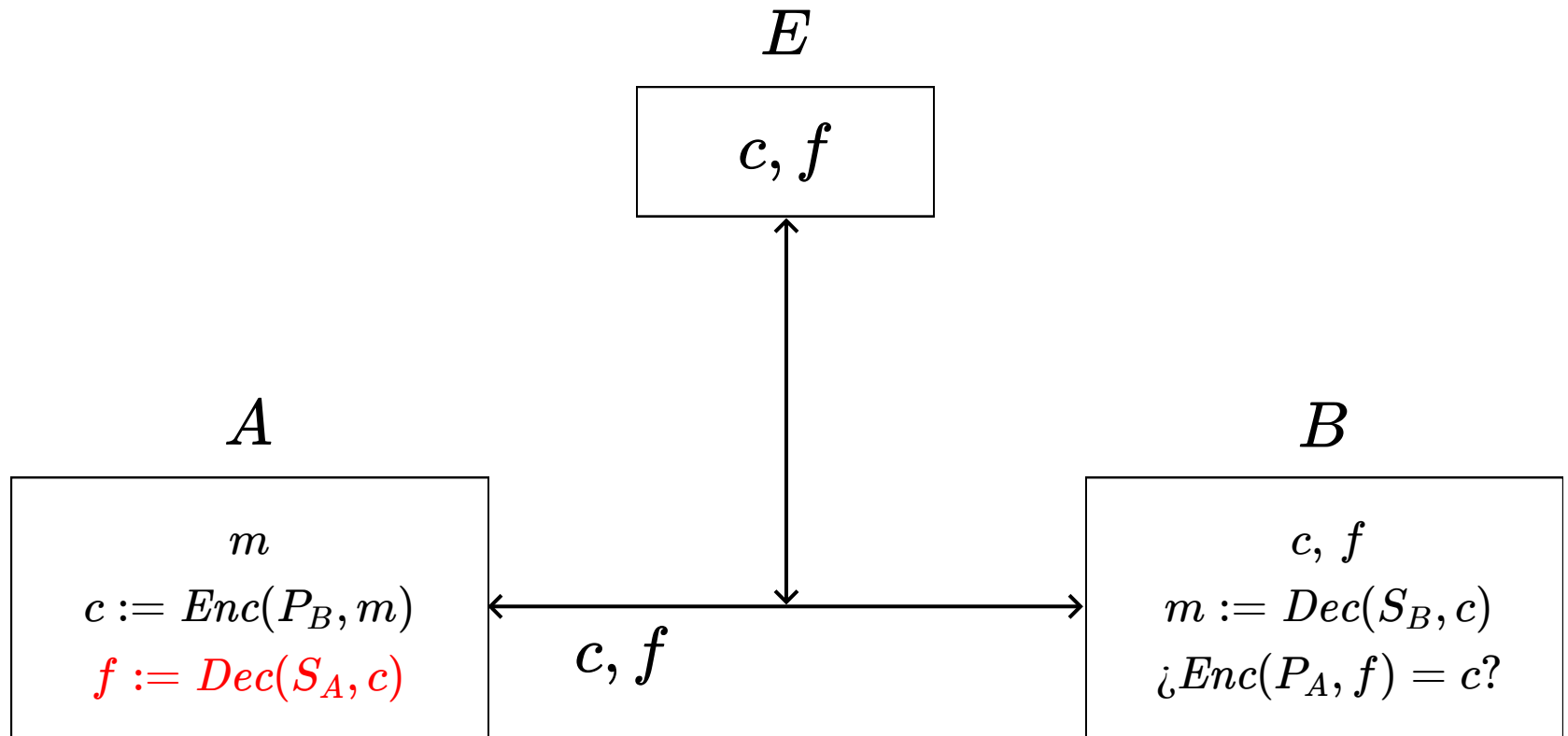
# El esquema de firmas digitales con RSA



# El esquema de firmas digitales con RSA



# $A$ puede firmar para $B$



# ¿Qué problema tiene el esquema anterior?

Firmar un mensaje  $m$  puede ser lento si  $m$  es un mensaje muy largo

Para solucionar este problema, se puede firmar  $h(m)$  en lugar de firmar  $m$ , donde  $h$  es una función de hash

# Firmas de Schnorr

Vamos a ver un segundo esquema para firmas digitales que está basado en el problema del logaritmo discreto

Se puede aplicar en cualquier grupo donde el problema de calcular el logaritmo discreto es difícil

# La definición de las firmas de Schnorr

Suponemos dado un grupo finito  $(G, *)$  y un elemento  $g \in G$  tal que  $|\langle g \rangle| = q$

- $G$ ,  $g$  y  $q$  son públicos
- Como vimos antes, se debe tener que  $|G|$  y  $q$  son números grandes

Además suponemos dada una función de hash  $h$

# La definición de las firmas de Schnorr

La clave privada de un usuario  $A$  es  $x \in \{1, \dots, q - 1\}$  y su clave pública es  $g^x$

# La definición de las firmas de Schnorr

La clave privada de un usuario  $A$  es  $x \in \{1, \dots, q - 1\}$  y su clave pública es  $g^x$

El usuario  $A$  quiere firmar un mensaje  $m$



# La definición de las firmas de Schnorr

A firma  $m$  de la siguiente forma:

1. Genera al azar  $k \in \{1, \dots, q - 1\}$  y calcula  $r = g^k$
2. Calcula  $v = h(r \| m)$  usando  $r$  como un string
3. Calcula  $s = k + v \cdot x$  interpretando  $v$  como un número natural
4. La firma de  $m$  es  $(v, s)$

# La verificación de una firma de Schnorr

Se puede verificar que  $(v, s)$  es una firma de  $m$  generada por  $A$  de la siguiente forma:

1. Calcule  $\alpha = g^s = g^{k+v \cdot x} = g^k * g^{v \cdot x} = g^k * (g^x)^v$
2. Calcule  $\beta = \alpha * ((g^x)^v)^{-1} = g^k * (g^x)^v * ((g^x)^v)^{-1} = g^k$
3. Verifique si  $v = h(\beta || m) = h(g^k || m)$

# Un ejemplo concreto: $\mathbb{Z}_p^*$ y SHA-256

Vamos a ver cómo se calculan las firmas de Schnorr considerando el grupo  $(\mathbb{Z}_p^*, \cdot)$  y la función de hash SHA-256

# El archivo grupo.txt

171254583176141379301960419792575778264088323240375085733932929816  
426671397476217788024387752387285929683446135893799323484756135034  
769321631669738132186983438164632891441853629126025225404949830905  
314972329658295365245072698488256583114202993359222957097432675083  
225259667739503949192575768420387716327420441424710535098501236058  
838158571626669177751934961573726561955583057270098912760065140004  
093658772181713883199238963093777917625906143118496429613802248519  
404604217104493689272529748703958739363879096722748832953774810081  
504758785902705917983505634881680809238046118223875201980540029906  
23911454389104774092183

# El archivo grupo.txt

804136732704618930269398466502670637484460828987437442572879766950  
943588145914066265021583283347132847033406462850869223199940184033  
204619256928735199168996327965689256248477327858420804098763156962  
852046406953236127404737444434499665183297937831884994374166211039  
599577842927081922243161092735600591383693246209977007623955404285  
528713802680696047027732622948281800396200445376440099579097404266  
367569212075872614586906123644389350913614794241444555184816239146  
854144435570778569782574185684916123388730701742837182360812569989  
290496084122159334449908899602188397218524185477760821259239701351  
0086894908468466292313  
  
637623513649726535646416995292055104892632668341827716175636313632  
77932854227

# Generación de claves públicas y privadas

```
1 def generar_clave_ElGamal():
2     f = open("grupo.txt", "r")
3     p = int(f.readline())
4     g = int(f.readline())
5     q = int(f.readline())
6     f.close()
7
8     x = random.randint(1, q - 1)
9     f = open("private_key.txt", "w")
10    f.write(str(x))
11    f.close()
12
13    f = open("public_key.txt", "w")
14    f.write(str(exp_mod(g, x, p)))
15    f.close()
```

# Cálculo de la firma

```
1 def firmar_Schnorr(mensaje: str) -> (int,int):
2     f = open("grupo.txt", "r")
3     p = int(f.readline())
4     g = int(f.readline())
5     q = int(f.readline())
6     f.close()
7
8     f = open("private_key.txt", "r") ← Clave privada
9     x = int(f.readline())
10    f.close()
11
12    k = random.randint(1, q - 1)
13    r = exp_mod(g,k,p)
14    hash = hashlib.sha256()
15    hash.update(str(r).encode() + mensaje.encode())
16    v = int(hash.hexdigest(),16)
17    s = k + v*x
18    return (v,s)
```

# Verificación de la firma

```
1 def verificar_firma_Schnorr(mensaje: str, firma: (int,int)) -> bool:
2     f = open("grupo.txt","r")
3     p = int(f.readline())
4     g = int(f.readline())
5     f.close()
6
7     f = open("public_key.txt","r") ← Clave privada
8     y = int(f.readline())
9     f.close()
10
11     alpha = exp_mod(g,firma[1],p)
12     beta = (alpha * inverso(exp_mod(y,v,p),p)) % p
13     hash = hashlib.sha256() ← Inverso en
14     hash.update(str(beta).encode() + mensaje.encode())      módulo p
15     return v == int(hash.hexdigest(),16)
```



# Utilizando la firma de Schnorr

```
1  if __name__ == "__main__":
2      generar_clave_ElGamal()
3      mensaje = "Alice transfiere 1000 BTC a Bob"
4      v,s = firmar_Schnorr(mensaje)
5      print("v: ",v)
6      print("s: ",s)
7      print(verificar_firma_Schnorr(mensaje, (v,s)))
8      print(verificar_firma_Schnorr(
9          "Alice transfiere 1001 BTC a Bob", (v,s)))
```

# Utilizando la firma de Schnorr

v: 19179042201810311532353596372007012380355747  
16208713513126393311491981373339

s: 12045255482702459011382783211874212188652965  
6054495776394483091481714436570025707060967347  
0109171054925710237068406591168928430281666737  
66562540691950833

True

False

# Utilizando la firma de Schnorr

private\_key.txt

6280425975373007028605265473256555979452984528  
8303070919508819505958485103115

# Utilizando la firma de Schnorr

public\_key.txt

665401503629675891132535192725194730008281568435527331916624507266  
720135320743020874962863575723769333563477742695123246920056083358  
270425409479177176872429154673331936673357019274805181347303525099  
421882694197793059997243313119399005264946335676130874750754292396  
017960534308852419191569421554144497668806746079951381281145191981  
298795380000674266218491737312936851588670945031223270377476741578  
827874990793196443213854054222841126339869690446992258786787897674  
688733496728258176595991785278698331955991491094941521791336272326  
587362125623511816512077582554153788080756547190301412898593008837  
8771090249201666151292

# ¿Qué ventajas tienen las firmas de Schnorr?

Son más pequeñas que *otras* firmas digitales

Son fáciles de combinar si varios usuarios deben firmar un mensaje

# ¿Cómo pueden firmar un documento dos usuarios?

Suponga que  $A$  y  $B$  deben firmar un mensaje  $m$

- Por ejemplo, un pago que debe ser autorizado por ambos usuarios

¿Cómo puede hacer esto usando RSA?

- ¿Es posible tener **una** clave pública para verificar que una firma es válida?

# Resolviendo el problema con firmas de Schnorr

Suponemos que:

- La clave privada de  $A$  es  $x_1$  y su clave pública es  $g^{x_1}$
- La clave privada de  $B$  es  $x_2$  y su clave pública es  $g^{x_2}$

La clave pública para verificar la firma de  $m$  por ambos usuarios es  $g^{x_1} * g^{x_2} = g^{x_1+x_2}$

# Resolviendo el problema con firmas de Schnorr

$A$  y  $B$  firman  $m$  de la siguiente forma:

1.  $A$  genera al azar  $k_1 \in \{1, \dots, q - 1\}$  y calcula  $r_1 = g^{k_1}$
2.  $B$  genera al azar  $k_2 \in \{1, \dots, q - 1\}$  y calcula  $r_2 = g^{k_2}$
3. Ambos calculan  $v = h((r_1 * r_2) || m)$  usando  $r_1 * r_2 = g^{k_1 + k_2}$  como un string



# Resolviendo el problema con firmas de Schnorr

4.  $A$  calcula  $s_1 = k_1 + v \cdot x_1$  interpretando  $v$  como un número natural
5.  $B$  calcula  $s_2 = k_2 + v \cdot x_2$  interpretando  $v$  como un número natural (de la misma forma que  $A$ )
6. Ambos calculan  $s = s_1 + s_2$ , y la firma de  $m$  es  $(v, s)$

# Resolviendo el problema con firmas de Schnorr

¿Cómo puede un usuario verificar que  $(v, s)$  es una firma de  $m$  generada por  $A$  y  $B$ ?

¿Cómo se puede generalizar esta idea para  $n$  usuarios?