

# Apendice\_ejemplo\_identificacion

May 24, 2021

## 0.1 Ejemplo de verificación del esquema MQ

*Autor:* Manuel Soto Jiménez

*Tutora:* María Ángeles Zurro Moro

En el siguiente notebook se muestra el código utilizado para generar ejemplos de verificación de identidad del esquema MQ, tal y como aparece en la Sección 4.2.1 de la memoria del trabajo. Recordemos la situación a simular:

- Un usuario, *Alice*, quien ha de identificarse, quiere demostrar el conocimiento de su clave privada  $\mathbf{s} \in \mathbb{F}^n$  sin, por supuesto, mostrar qué clave es.
- Un segundo usuario, *Bob*, la parte verificante, quiere comprobar que en efecto se comunica con Alice a partir de pedirle uno de entre tres *challenges*  $c_i$  que consistirán en comprobaciones aritméticas con  $\mathcal{P}$ ,  $\mathcal{P}(\mathbf{s})$  y una serie de valores  $\mathbf{r}_0, \mathbf{r}_1, \mathbf{t}_0, \mathbf{t}_1 \in \mathbb{F}^n$ ,  $\mathbf{e}_0, \mathbf{e}_1 \in \mathbb{F}^m$ ; cuya obtención ha sido explicada a lo largo de la memoria.

Definamos primero nuestros parámetros con respecto a dimensiones y órdenes de los anillos sobre los que trabajar.

```
[23]: q = 4  
      n = 4  
      m = 6
```

Creamos con ello el cuerpo  $\mathbb{F}_q$  y los anillos polinomiales del espacio de salida y llegada. Hacemos uso de `inject_variables()` para establecer las  $x_i$  sobre las que se computa.

```
[24]: K.<alpha> = FiniteField(q)  
      F_n = PolynomialRing(K, 'x_', n, order="lex")  
      F_m = PolynomialRing(K, 'x_', m, order="lex")  
  
      F_n.inject_variables()  
      F_m.inject_variables()
```

Defining  $x_0, x_1, x_2, x_3$

Defining  $x_0, x_1, x_2, x_3, x_4, x_5$

A continuación generamos aleatoriamente  $\mathcal{P}$ . Realmente este paso no sería aleatorio, sino que esta función consistiría en la clave pública de *Alice*, con clave privada  $\mathcal{P}^{-1} = \mathcal{T}^{-1}(\mathcal{F}^{-1}(\mathcal{S}^{-1}))$ . Eliminamos términos constantes de cada polinomio para simplificar cálculos de la forma polar  $\mathcal{G}$ .

[25]: *# los hacemos sin terminos constantes para simplicidad en la forma polar*

```
p_prev = [F_n.random_element(degree=2) for _ in range(m)]
p = [f - f.constant_coefficient() for f in p_prev]

print(latex(p))
```

```
\left[\alpha x_{0} x_{3} + \alpha x_{0} + \left(\alpha + 1\right) x_{2},
\left(\alpha + 1\right) x_{1} x_{2} + \left(\alpha + 1\right) x_{2}^2 +
\left(\alpha + 1\right) x_{3}^2, \alpha x_{1}^2 + \left(\alpha + 1\right)
x_{2}^2 + \alpha x_{3}, \left(\alpha + 1\right) x_{1} x_{2} + \alpha x_{2}^2
+ x_{3}^2, \alpha x_{1}^2 + \alpha x_{1} x_{2} + \alpha x_{2} x_{3} + \alpha
x_{2}, \alpha x_{0} x_{1} + x_{0} x_{2} + x_{0} x_{3} + \alpha x_{3}^2\right]
```

Obtenemos a continuación una clave privada  $s$  y calculamos  $\mathbf{v} = \mathcal{P}(s)$ .

```
[26]: s = [K.random_element() for _ in range(n)]
print(s)
v = [f(s) for f in p]
print(v)
```

```
[alpha + 1, 0, 1, alpha]
[0, 1, 0, 1, 1, alpha + 1]
```

Los siguientes valores recordemos que serán generados aleatoriamente para cada ronda de identificación entre Alice y Bob:  $\mathbf{r}_0, \mathbf{t}_0 \in \mathbb{F}^n$ ,  $\mathbf{e}_0 \in \mathbb{F}^m$ .

```
[27]: r_0 = [K.random_element() for _ in range(n)]
t_0 = [K.random_element() for _ in range(n)]
e_0 = [K.random_element() for _ in range(m)]

print((r_0,t_0,e_0))
```

```
([alpha, alpha + 1, alpha + 1, alpha + 1], [alpha + 1, 1, 1, alpha + 1], [0,
alpha + 1, alpha + 1, 0, 0, 0])
```

A continuación calculamos  $\mathbf{r}_1, \mathbf{t}_1 \in \mathbb{F}^n$ ,  $\mathbf{e}_1 \in \mathbb{F}^m$  a partir de los tres valores anteriores junto a  $s$  y  $\mathcal{P}$ .

```
[28]: r_1 = [s[i] - r_0[i] for i in range(n)]
t_1 = [r_0[i] - t_0[i] for i in range(n)]
e_1 = [p[i](r_0) - e_0[i] for i in range(m)]

print((r_1,t_1,e_1))
```

```
([1, alpha + 1, alpha, 1], [1, alpha, alpha, 0], [alpha + 1, alpha, 0, 0, alpha,
1])
```

A continuación definimos la forma polar  $\mathcal{G}$  a partir de  $\mathcal{P}$  y dos vectores de  $\mathbb{F}^n$ . Recordemos cómo se define  $\mathcal{G}$  para polinomios homogéneos:

$$\mathcal{G}(\mathbf{x}, \mathbf{y}) = \mathcal{P}(\mathbf{x} + \mathbf{y}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{y}).$$

```
[29]: def g(v1,v2):
        return [p[i]([v1[j]+v2[j] for j in range(len(v1))]) - p[i](v1) - p[i](v2) for i in range(len(p))]
```

Mediante la forma polar de  $\mathcal{P}$  junto a las seis variables anteriores, *Alice* crea los tres *challenges* que demuestran que conoce  $\mathbf{s}$ . Dichos *challenges* dependerán de una función de *commitment*  $Com(x, y)$ , que ha de ser resistente a colisiones y preimágenes. Para simplificar y verificar los resultados, tomamos  $Com = Id$ . El lector de este notebook podrá cambiar la función  $com(x, y)$  como desee, pero es evidente que, si  $x = x', y = y'$ , entonces  $Com(x, y) = Com(x', y')$  independientemente de la función *hash* escogida.

```
[30]: def com(x,y):
        return x,y

c_0 = com(r_1 , [g(t_0,r_1)[i] + e_0[i] for i in range(m)])
c_1 = com(t_0, e_0)
c_2 = com(t_1, e_1)
print((c_0,c_1,c_2))
```

```
(([1, alpha + 1, alpha, 1], [0, 0, alpha + 1, alpha + 1, alpha, 1]), ([alpha + 1, 1, 1, alpha + 1], [0, alpha + 1, alpha + 1, 0, 0, 0]), ([1, alpha, alpha, 0], [alpha + 1, alpha, 0, 0, alpha, 1]))
```

Por último, en la siguiente celda nos situamos desde el punto de vista de *Bob* y realizamos la comprobación pertinente en función del *challenge* elegido, a partir del cual *Alice* nos proveerá de la información necesaria. El lector puede desde este notebook asignar a *ch* un valor entre 0, 1 y 2 para elegir qué *challenge* comprobar.

```
[34]: ch = 1 # ch in {0,1,2}

if ch==0:
    c1_0 = com([r_0[i] - t_1[i] for i in range(n)], [p[i](r_0) - e_1[i] for i in range(m)])
    c2_0 = com(t_1, e_1)
    print(c1_0,c2_0)
    print("Comprobaciones ch = 0:")
    print("c_1: " + str(c1_0==c_1) + "\tc_2: " + str(c2_0 == c_2))
elif ch==1:
    c0_1 = com(r_1,[v[i] - p[i](r_1) - g(t_1,r_1)[i] - e_1[i] for i in range(m)])
    c2_1 = com(t_1, e_1)
    print(c0_1,c2_1)
    print("Comprobaciones ch = 1:")
    print("c_0: " + str(c0_1==c_0) + "\tc_2: " + str(c2_1 == c_2))
elif ch==2:
```

```

c0_2 = com(r_1, [g(t_0,r_1)[i] + e_0[i] for i in range(m)])
c1_2 = com(t_0, e_0)
print(c0_2,c1_2)
print("Comprobaciones ch = 2:")
print("c_0: " + str(c0_2==c_0) + "\tc_1: " + str(c1_2 == c_1))
else:
    print("ch invalida")

```

```

([1, alpha + 1, alpha, 1], [0, 0, alpha + 1, alpha + 1, alpha, 1]), ([1, alpha,
alpha, 0], [alpha + 1, alpha, 0, 0, alpha, 1]))

```

Comprobaciones ch = 1:

```

c_0: True      c_2: True

```