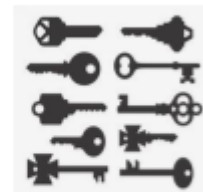


# Estudio e implementación de esquemas de compartición de secretos.

Compartición de secretos de manera infinita.



...

*Manuel Jose Mora Codero.  
Criptografía y Criptoanálisis.  
Máster en Ciberseguridad y Privacidad URJC 2021-2022*

*Diciembre 2022.*

# Introducción y Contexto.

Un esquema de compartición de secretos es un sistema que permite repartir información sobre un secreto entre diferentes participantes, de manera que solo cuando un determinado número de esas partes se juntan, es posible recuperar el secreto.

Ese número mínimo de partes necesarias para recuperar el secreto se conoce con el término en inglés de **threshold** y se suele denotar con la letra **k**.

Al número de *participantes* o **parties** se suele denotar por **n**.

El *secreto* o **secret** se suele denotar por **s** y se suele indicar como un conjunto de bits de *longitud l*.

Por último tenemos el **share** o fragmento del secreto que recibe cada participante, que no revela información alguna sobre el secreto y que permite reconstruir el secreto cuando se reúne un conjunto autorizado de participantes.

Los esquemas de compartición de secretos fueron inventados de manera independiente por Adi Shamir y George Blakley en 1979.

## Importancia.

Los sistemas tradicionales de encriptación no permiten altos niveles de confidencialidad y confiabilidad simultáneamente. Si un secreto es encriptado y almacenado en un único lugar en una única copia, presentará un alto grado de confidencialidad, pero si este secreto es almacenado con diferentes copias en diferentes lugares, se vuelve más confiable, pero menos confidencial al exponer más vectores de ataque.

Los esquemas de compartición de secretos permiten alcanzar mejores niveles de confidencialidad y confiabilidad de manera simultánea.

Estos esquemas de compartición de secretos son utilizados en sistemas de computación distribuida, cloud computing, sistemas de votación y subasta, etc.

\*[https://hmong.es/wiki/Secret\\_sharing](https://hmong.es/wiki/Secret_sharing)

# Motivación.

La compartición de secretos cobra importancia dentro de la Criptografía con el crecimiento de la computación distribuida y en la nube, algo que está ya muy integrado en el mundo tecnológico y empresarial, y con el crecimiento del uso de las criptomonedas.

Los sistemas de compartición de secretos tradicionales más utilizados, como por ejemplo el sistema de *Shamir*, tiene la limitación de que un límite superior de participantes ha de ser conocido previamente, es decir, el valor de  $n$  se ha de fijar para disponer del esquema de compartición de secretos.

En el paper "*How to Share a Secret, Infinitely*", fechado en 2016, *Ilan Komargodski, Moni Naor y Eylon Yogev* proponen el concepto de *Evolving K-threshold Scheme* y plantean varias formas de construcción de un esquema de compartición de secretos donde el número  $n$  de participantes no es necesario que sea fijado.

En este documento también se trata el tamaño del secreto a compartir, algo que es importante considerar.

\*"*How to Share a Secret, Infinitely*" : <https://eprint.iacr.org/2016/194.pdf>

# Objetivo.

El objetivo de este Trabajo Fin de Máster no es el de plasmar la matemática teórica que se cuenta en el mencionado documento, sino el de entender y plasmar de una manera más cercana su contenido, las soluciones al problema que se plantean y elaborar de manera práctica una prueba de concepto de alguna de estas soluciones.

# El Esquema de Compartición de Secretos de Shamir.

Es el esquema de compartición de secretos más común y usado actualmente.

Este esquema está basado en el teorema matemático de interpolación de Lagrange, según el cual  $k$  puntos son necesarios para definir un determinado polinomio de grado  $k-1$ , es decir, con menos de  $k$  puntos será matemáticamente imposible re-construir el polinomio de grado  $k-1$  generado a partir de esos  $k$  puntos.

Este es el principio matemático que indica que 2 puntos son necesarios para construir una determinada recta, 3 puntos para reconstruir una parábola, etc.

Este  $k$  sería el *threshold* del esquema, es decir, el número mínimo de partes necesarias para poder reconstruir el polinomio que es la llave al secreto  $s$ . Y las partes serán los puntos que cumplen con la ecuación que define el polinomio de grado  $k-1$ .

En la entrada en Wikipedia sobre el esquema de Shamir se explica bastante bien su funcionamiento.

En este enlace también se explica porque este esquema deja de ser seguro si no se aplica sobre un cuerpo finito. Al deberse aplicar sobre un cuerpo finito, normalmente  $\mathbb{Z}_p$  con  $p$  primo, implica pues que debemos fijar un  $p$  primo mayor que el número de partes  $n$ , por lo que tenemos que conocer previamente  $n$ , el número de partes.

Se puede consultar todo esto en el siguiente enlace:

\*Enlace Wikipedia Shamir Secret Share Scheme: [https://en.wikipedia.org/wiki/Shamir%27s\\_Secret\\_Sharing](https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing)

El tamaño del share enviado a cada Party en este esquema es

**$\text{Size}(\text{share}[P_i]) = \log_2(p) \rightarrow \text{aprox } \log_2(n)$** ,  $P_i$  participante, donde  $i \in \{1, 2, \dots, n\}$

es decir, crece con  $n$  el número de *Parties*.

El esquema de compartición de secretos de Shamir no permite crecer al número de participantes de manera indefinida, pero es el esquemas más común, existen ya numerosas librerías que lo implementan en diferentes lenguajes de programación y en nuestro caso lo vamos a usar dentro de las soluciones propuestas sobre el problema de compartición de secretos de manera indefinida.

Denotaremos por **SSS(k, n)** al esquema de Shamir de compartición de secretos con threshold  $k$  y número de participantes  $n$ .

## Aproximaciones a la solución para la compartición infinita.

Introducción al método ideado para la gestión infinita de participantes: Dimensional Reduction.

El concepto principal que se maneja en este paper para gestionar la llegada de participantes al esquema, llamado “Dimensional Reduction”, es el de **generación**, que es una agrupación finita de participantes.

- Cada Participante  $P$  pertenece a una generación  $g_i$ .
- Dada un  $k$  threshold fijo, la generación  $g_i$  contiene  $i \cdot 2^{(k-1)}$  Participantes.
- De esta manera, las generaciones se van completando según van llegando los Participantes:

si $k = 2$	
$g_1 = \{P_1, P_2\}$	<- $1 \cdot 2^1 = 2$ elementos
$g_2 = \{P_3, P_4, P_5, P_6\}$	<- $2 \cdot 2^1 = 4$ elementos
$g_3 = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}\}$	<- $3 \cdot 2^1 = 6$ elementos
...	

De esta manera los Participantes van llegando de una manera infinita, siendo agrupados en Generaciones finitas.

### Primera aproximación a la resolución del problema de la compartición de secretos de manera infinita.

#### Evolving 2-threshold Scheme.

El documento de *Ilan Komargodski, Moni Naor y Eylon Yogev* propone en un ejemplo sencillo fijando el threshold  $k$  igual a 2, cómo sería posible construir un esquema de compartición de secretos de tipo evolving.

Básicamente este ejemplo consiste en:

- 1) Implementar un esquema de compartición con  $k=2$  como *threshold* y  $n$  igual al tamaño de cada generación como forma de compartir shares dentro de una misma generación. Se puede implementar con un esquema de compartición de Shamir  $SSS(k=2, n=size(g_i))$ .
- 2) Para el caso en el que participantes de diferentes generaciones quieran reconstruir el secreto, se puede en este ejemplo más simple donde  $k=2$  partir de una solución en la que para cada generación se usa una cadena de bits  $b$  aleatoria de la misma longitud que el secreto  $s$  convertidos a bits, e ir ampliando suma directa del secreto  $s$  y  $b$  de la siguiente manera:

Supongamos que tenemos 2 generaciones,

$g_1 = \{P_1, P_2\}$   
 $g_2 = \{P_3, P_4, P_5, P_6\}$

convertimos el secreto  $s$  en su cadena de bits. Generamos  $b_1$  y  $b_2$  dos cadenas de bits aleatorios de la misma longitud que  $s$ , y repartimos entre cada participante de la siguiente manera:

$[b_1] \rightarrow$  Participantes  $P_1$  y  $P_2$   
 $[b_2, s+b_1] \rightarrow$  Participantes  $P_3, P_4, P_5$  y  $P_6$   
(Nota: aquí la suma + indica suma directa de bits)

De esta manera si dos participantes de diferentes generaciones, por ejemplo  $P_2$  y  $P_4$ , necesitan reconstruir el secreto, mediante la suma directa  $b_1 + (s + b_1)$ , obtendrían el valor de  $s$ .

Si aparecen nuevos participantes y se abre una nueva generación  $g_3$ ,

$g_3 = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}\}$

Entonces a estos nuevos participantes se les repartiría:

$[b_3, s+b_1, s+b_2] \rightarrow P_7, P_8, P_9, P_{10}, P_{11}$  y  $P_{12}$

y de idéntica manera, mediante suma directa, dos participantes de diferentes generaciones podrían reconstruir el secreto  $s$ .

En este trabajo he querido implementar un método alternativo que permita no fijar un valor de  $k$  *threshold*, y que mediante la agrupación de participantes en generaciones, permita también que el número de participantes vaya creciendo de manera indefinida.

En este caso, el modelo implementado, necesitará, a diferencia del ejemplo anterior, repartir nuevos shares a generaciones ya completas cada vez que aparezca una nueva generación.

Este modelo consiste en utilizar sistemas de compartición de secretos para un número de participantes finito de la siguiente manera:

1) Dentro de una misma generación:

Sabemos que dentro de una generación el número de participantes es finito, por que lo que dentro de cada generación se impronta un **SSS(k, n\_i)** donde **n\_i** es el número de Participantes dentro de la generación **g\_i**.

Por ejemplo, siguiente el ejemplo anterior donde  $k = 2$ ,

$$g_2 = \{P_3, P_4, P_5, P_6\} \leftarrow \text{SSS}(2, 4)$$

En este caso si tamaño del share sería,

$$\text{size}(P_i) \leq \log_2(i) + k$$

2) Entre Participantes de diferentes generaciones:

Vamos a suponer que  $K = 3$ , lo vamos a representar el caso de dos generaciones,  $g_1$  y  $g_2$ .

$$g_1 = \{P_1, P_2, P_3, P_4\}$$

$$g_2 = \{P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}\}$$

Cuando se completa una nueva generación, en este caso la generación  $g_2$ , generamos

$n^\circ$  generaciones  $\cdot (k - 1)$  shares con threshold igual a  $k$  del secreto  $S$ , en nuestro caso, al ser generaciones,

$$[s1_1, s1_2, s2_1, s2_2], 2 \cdot (3 - 1) \text{ shares}$$

De los cuales,

1) los  $k-1$  primeros,  $(s1_1, s1_2)$ , se repartirán entre los participantes de  $g_1$  de la siguiente manera:

- $\text{SSS}(\text{key} = s1_1, \text{threshold } k = 1, n = \text{size}(g_1))$
- $\text{SSS}(\text{key} = s1_2, \text{threshold } k = 2, n = \text{size}(g_1))$

2) los  $k-2$  siguientes,  $(s2\_1, s2\_2)$ , se repartirán entre los participantes de  $g\_2$  de manera similar.

- $SSS(\text{key} = s2\_1, \text{threshold } k=1, n = \text{size}(g\_2))$
- $SSS(\text{key} = s2\_2, \text{threshold } k=2, n = \text{size}(g\_2))$

Si apareciese una nueva generación  $g3$  de 16 elementos,

$g3 = \{P_{13}, \dots, P_{29}\}$

Entonces deberemos generar de nuevo  $3*(k-1) = 6$  shares de  $S$  con threshold  $k = 2$

$[s1\_1, s1\_2, s2\_1, s2\_2, s3\_1, s3\_2]$

y repartimos  $s1\_1$  y  $s1\_2$  en  $g\_1$  de la siguiente manera:

- $SSS(\text{key} = s1\_1, \text{threshold } k=1, n = \text{size}(g\_1))$
- $SSS(\text{key} = s1\_2, \text{threshold } k=2, n = \text{size}(g\_1))$

repartimos  $s2\_1$  y  $s2\_2$  en  $g2$  de la siguiente manera:

- $SSS(\text{key} = s2\_1, \text{threshold } k=1, n = \text{size}(g\_2))$
- $SSS(\text{key} = s2\_2, \text{threshold } k=2, n = \text{size}(g\_2))$

y  $s3\_1$  y  $s3\_2$  en  $g3$  de la siguiente manera:

- $SSS(\text{key} = s3\_1, \text{threshold } k=1, n = \text{size}(g\_1))$
- $SSS(\text{key} = s3\_2, \text{threshold } k=2, n = \text{size}(g\_1))$

De esta manera, si se juntan 3 participantes de diferentes generaciones, serán capaces de recuperar el secreto  $S$ .

Por ejemplo si se juntan 2 participantes de  $g1$  será capaces de recuperar:

$s1\_1$  ya que este se repartió en  $g1$  con un threshold igual a 1.

$s1\_2$  ya que este se repartió en  $g1$  con un threshold igual a 2.

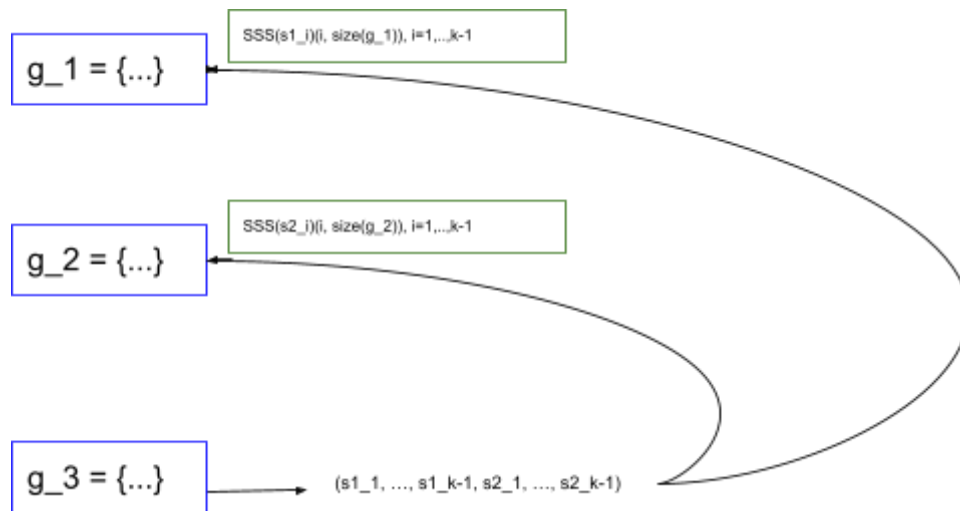
y si estos a su vez se juntan con un tercero de  $g3$ , este podrá recuperar

$s3\_1$  ya que este se repartió en  $g3$  con un threshold igual a 1.

Así que estos tres participantes de diferentes generaciones con  $s1\_1$ ,  $s1\_2$  y  $s3\_1$  serían capaces de reconstruir el secreto  $S$ .



De manera gráfica,



Actuando de esta manera, el número de Participantes puede crecer de manera indefinida.

El tamaño del share para la compartición entre diferentes generaciones se estima en

$$size(P\_i) = \max \{ \log_2(size(g\_i)), |s\_i| \}$$

Lo que aproxima al tamaño total del share cuando aparece una nueva generación de

$$size(P\_i) \leq \log_2(i) + (k-1) * (\log_2(t) + k)$$

Es objetivo de este trabajo implementar una prueba de concepto que permita comprobar que esta aproximación es correcta, y que pueda servir de punto de partida para una solución más general. No es objetivo de este trabajo una solución funcional, robusta y completa.

A continuación, mostramos esta prueba de prueba de concepto, donde ubicarla, que necesitaríamos para poder ejecutarla, así como una visualización de una ejecución de la misma.

Este software se ha desarrollado con Python 3.8.8, donde se han usado principalmente la siguiente librería:

- shamirs-2.0.2  
(<https://pypi.org/project/shamirs/>)

Como algoritmo de compartición de secretos estándar usado para la compartición de secretos para un número fijo de participantes, en una misma generación y entre generaciones.

El código de esta PoC se encuentra disponible en el siguiente repositorio:

[https://github.com/MCYP-UniversidadReyJuanCarlos/21-22\\_mamoco](https://github.com/MCYP-UniversidadReyJuanCarlos/21-22_mamoco)

Se introduce la clave como un entero positivo:

```
$ python sss_infinitly.py
Por favor, introduce la clave como un número entero positivo:
█
```

Introducimos el Secreto o clave y el threshold:

El threshold, en este ejemplo sería de 2 participantes.

```
$ python sss_infinitly.py
Por favor, introduce la clave como un número entero positivo:
343535345
Seguimos...
Ahora introduce el k-threshold como un numero entero positivo:
2
Seguimos...

*****
Por favor, elige una opción para continuar:

1 - Añadir un nuevo Participante.

2 - Pasar un lista de participantes para tratar de desencriptar la clave.

3 - Finalizar.

4- Visualizar la lista de Participantes.

5 - Visualizar la lista de Participantes que desean desencriptar.

6 - Desencriptar.

7 - Visualizar participantes y generaciones.

*****
█
```

Vamos introduciendo participantes con la opción 1, escribiendo sus nombres hasta tener una lista:

```
*****
Por favor, elige una opción para continuar:

1 - Añadir un nuevo Participante.

2 - Pasar un lista de participantes para tratar de desencriptar la clave.

3 - Finalizar.

4- Visualizar la lista de Participantes.

5 - Visualizar la lista de Participantes que desean desencriptar.

6 - Desencriptar.

7 - Visualizar participantes y generaciones.

*****
4

['Manuel', 'Jose', 'Maria', 'Ana', 'Luis', 'Roberto']
```

Y llegado un momento añadimos una lista de nombres separados por coma de participantes que desean revelar,

```
*****
Por favor, elige una opción para continuar:

1 - Añadir un nuevo Participante.

2 - Pasar un lista de participantes para tratar de desencriptar la clave.

3 - Finalizar.

4- Visualizar la lista de Participantes.

5 - Visualizar la lista de Participantes que desean desencriptar.

6 - Desencriptar.

7 - Visualizar participantes y generaciones.

*****
2
Introduce una lista con los nombres de los participantes separados por una coma.
Por Ejemplo: Pedro, Luis, Manuel
Jose,Roberto█
```

Y comprobamos si es posible revelar la clave introducida al inicio:

```
*****
Por favor, elige una opción para continuar:

1 - Añadir un nuevo Participante.

2 - Pasar un lista de participantes para tratar de desenscriptar la clave.

3 - Finalizar.

4- Visualizar la lista de Participantes.

5 - Visualizar la lista de Participantes que desean desenscriptar.

6 - Desenscriptar.

7 - Visualizar participantes y generaciones.

*****
6
...
Numero de generaciones: 2
Los candidatos son de diferentes generaciones
La clave es:
.
..
...
-----
343535345
-----
```

## Segunda aproximación a la resolución del problema de la compartición de secretos de manera infinita.

En el documento que da lugar a este trabajo, se explica una aproximación un tanto diferente, pero también basada en la agrupación de participantes en generaciones.

Esta aproximación se centra más en el número de participantes de cada generación que se presentan para revelar el secreto.

Lo mejor es explicar esta aproximación mediante un ejemplo sencillo.

**S** es el secreto y pertenece a  $\{0, 1\}^{long}$ , es decir es un conjunto de bits, 0's y 1's, de longitud long.

En este ejemplo el cada generación está compuesta  $(k - 1) * (k^g)$  participantes, es decir, si basamos el ejemplo en dos generaciones,

$g0 = \{P_1, \dots, P_{k-1}\}$   
 $g1 = \{P_k, \dots, P_{k^2-1}\}$   
...

Se define **A** como el conjunto

$A = (c_0, c_1, \dots, c_i)$ , donde

$c_0$  sería el numero participantes de  $g0$  que se postulan para relevar el secreto,  
 $c_1$  sería el numero participantes de  $g1$  que se postulan para relevar el secreto,  
etc.

**A** sería como un indicador de estado, donde llevar la cuenta de participantes de cada generación que se postulan para revelar el secreto.

Se define **S\_A** como el secreto al intentar ser revelado en el estado  $A$ , es decir, un estado de encriptación del secreto  $S$  dado un momento concreto en el que  $A$  tiene los valores del número de participantes por generación que se postulan para revelar el secreto en dicho momento.

Definimos como **S\_prevA** al estado  $S_A$  previo.

Al empezar, antes de nada se fija **S\_prevA = S** como el estado inicial.

Los participantes van llegando y van formando parte de una u otra generación en función del orden en el que van llegando.

Cada vez que llega un participante al esquema, se actualiza el estado  $A$  y se suman todos los valores del estado en ese momento, y si

1)

$$c_0 + c_1 + \dots < k$$

se elige un valor random  $r$  perteneciente a  $\{0, 1\}^{long}$ , y se actualiza  $S_A$

$$S_A = S_{prevA} + r \quad (* \text{ la suma es suma directa})$$

y el valor  $r$  se reparte en las generaciones existentes mediante un esquem de compartición

$$\begin{aligned} &SSS(key = r, k = c_0, n = size(g0)) \text{ para la generación } g0, \\ &SSS(key = r, k = c_1, n = size(g1)) \text{ para la generación } g1, \end{aligned}$$

y así para todas las generaciones que existan en ese momento.

De esta manera  $c_0$  miembros de  $g0$  serían capaces de recuperar  $r$ ,  
y  $c_1$  miembros de  $g1$  también serían capaces de recuperar  $r$ , etc.

2) Cuando,

$$c_0 + c_1 + \dots = k \text{ (threshold)}$$

Entonces tomamos

$S_A$ , que será igual a  $S + r_1 + r_2 + \dots$  (Suma directa)

y lo repartimos entre las generaciones existentes usando un esquema de partición de secretos,

$SSS(\text{key} = S_A, k = c_i, n = \text{size}(g_i))$  con  $i=0, 1, 2, \dots$  con  $i$  indicador de cada generación.

De esta manera,

- 1) Si  $k$  miembros de una misma generación se postulan para revelar el secreto  $S$ , estos  $c_i$  miembros de  $g_i$  pueden recuperar

- $S + r_0 + r_1 + \dots$
- $r_0$ ,
- $r_1$ ,
- ...

De manera que sumando todos con suma directa, obtendrían el valor de  $S$ .

- 2) Y si  $k$  miembros de diferentes generaciones se postulasen,

supongamos que  $c_0 + c_1 + \dots + c_i = k$

$c_0$  miembros de  $g_0$  recuperarán  $r_0$

$c_1$  miembros de  $g_1$  recuperarán  $r_1$

...

$c_i$  miembros de  $g_i$  recuperarán  $S + r_0 + r_1 + \dots$

e igualmente, vía suma directa, serían capaces de recuperar el secreto  $S$ .

## Conclusiones y trabajo futuro.

Como conclusiones a este trabajo destacaría la propuesta de este paper sobre cómo abordar el complejo caso de no disponer de un número  $n$  fijo de participantes en un esquema de compartición de secretos. Es una manera ingeniosa la de ir creando conjuntos finitos de participantes, las generaciones, y aplicar entre ellas un modelo de compartición de secretos que permita dicha compartición entre grupos y dentro del propio grupo.

El número de partes a repartir aumenta no solo con la llegada de nuevos participantes, sino también según aparecen nuevas generaciones, pero permite no fijar de inicio el número máximo de participantes, una característica importante y necesaria para un gran número de casos de uso, ahora que la computación cloud está tan extendida.

La implementación de la prueba de concepto en Python ha sido buena idea, ya que hay ya librerías que permiten implementar un esquema de Shamir fácilmente y operar también fácilmente con listas donde almacenar las partes para cada participante.

Como trabajo futuro, quedaría la implementación de la segunda manera de gestión de shares para un número no fijo de participantes basada en contabilizar el número de participantes de cada generación que desean revelar el secreto, y realizar un análisis de rendimiento computacional entre ambas soluciones.

## Bibliografía y enlaces consultados.

[1] <https://www.cs.bgu.ac.il/~beimel/Papers/Survey.pdf>

[2] [https://hmong.es/wiki/Secret\\_sharing](https://hmong.es/wiki/Secret_sharing)

[3] [https://en.wikipedia.org/wiki/Shamir%27s\\_Secret\\_Sharing](https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing)

[4] [https://github.com/MCYP-UniversidadReyJuanCarlos/21-22\\_mamoco](https://github.com/MCYP-UniversidadReyJuanCarlos/21-22_mamoco)