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*.

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 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

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 sistemas 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 Zp 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

El tamaño de cara Party en este esquema es

$$\textbf{Size}(P_i) = \log_2(p) \rightarrow \textbf{aprox } \log_2(n), \quad P_i \text{ participante, donde } i \text{ en } \{1,2,....,n\}$$

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*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*2^1 = 2 elementos
g_2 = {P_3, P_4, P_5, P_6} <- 2*2^1 = 4 elementos
g_3 = {P_7, P_8, P_9, P_10, P_11, P_12} <- 3*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 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 mismas longitud que s, y repartimos entre cada participante de la siguiente manera:

   [b_1] → Participantes P_1 y P_2
   [b_2 , s+b_1] → 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:

   [b3, s+b1, s+b2] → 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 imprenta 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} <- SSS(2, 4)

En este caso si tamaño del share sería,

size(P_i) <= 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

nº generaciones * (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* (3 -1) 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:

- SSS(key = s1_1, threshold k= 1, n = size(g_1))
- SSS(key = s1_2, threshold k= 2, n = size(g_1))

2) los k -2 siguientes, (s2_1, s2_2), se repartirán entre los participantes de g_2 de manera similar.

- SSS(key = s2_1, threshold k= 1, n = size(g_2))
- SSS(key = s2_2, threshold k= 2, n = size(g_2))

Si apareciese una nueva generación g3 de 16 elementos,

g3 = {P_13, …, 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(key = s1_1, threshold k= 1, n = size(g_1))
- SSS(key = s1_2, threshold k= 2, n = size(g_1))

repartimos s2_1 y s2_2 en g2 de la siguiente manera:

- SSS(key = s2_1, threshold k= 1, n = size(g_2))
- SSS(key = s2_2, threshold k= 2, n = size(g_2))

y s3_1 y s3_2 en g3 de la siguiente manera:

- SSS(key = s3_1, threshold k= 1, n = size(g_1))
- SSS(key = s3_2, threshold k= 2, n = 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:

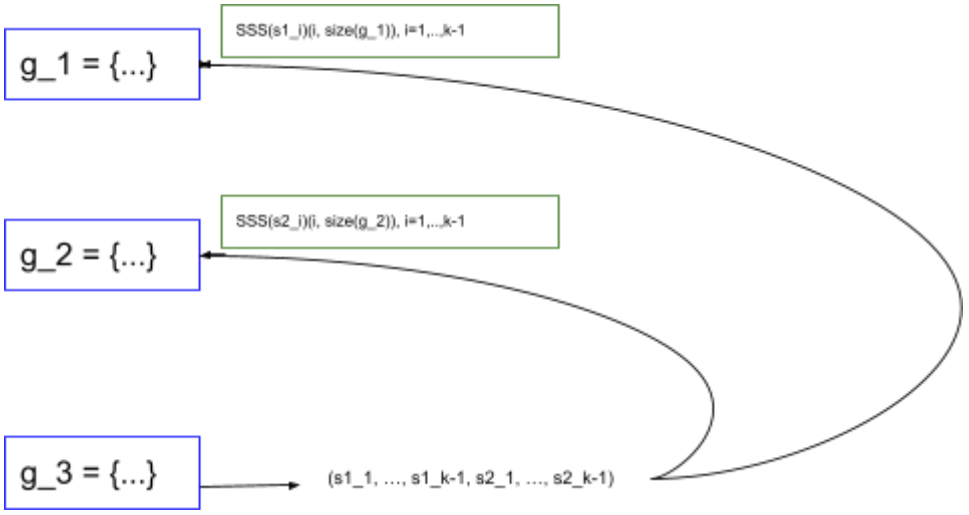
s1_1 ya que este se repartió en g1 con un threshold igual a 1.
s1_2 ya que este se repartió en g1 con un threshold igual a 2.

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) <= 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:

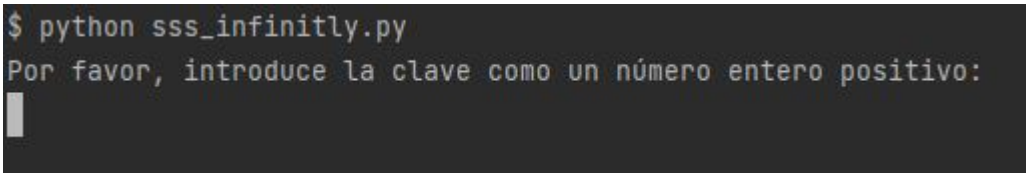
- shamirs-2.0.2
  (https://pypi.org/project/shamirs/)

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:

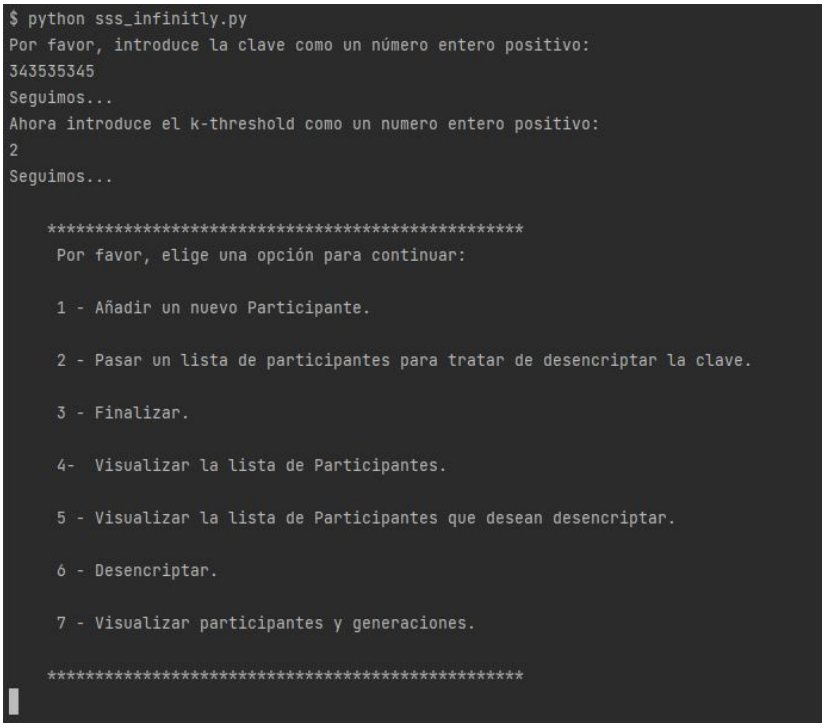
https://github.com/ManuelJoseMora/SecretSharingInfinitly

Se introduce la clave como un entero positivo:

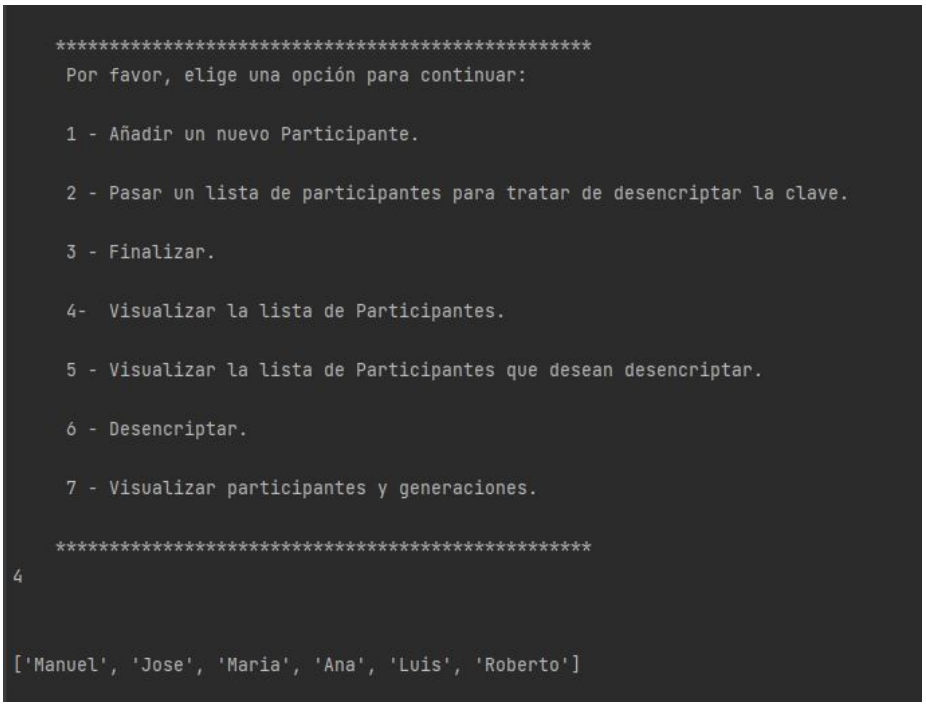


Introducimos el Secreto o clave y el threshold:

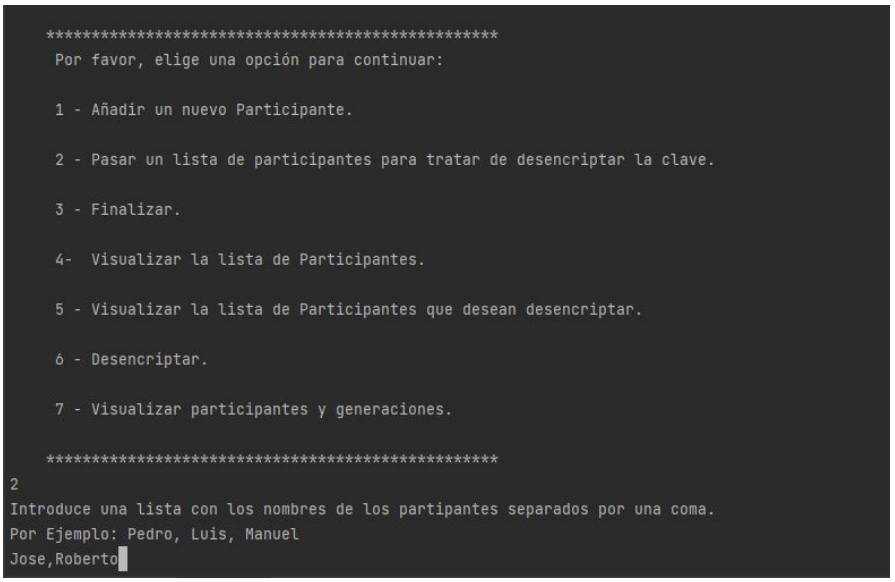
El threshold, en este ejemplo sería de 2 participantes.



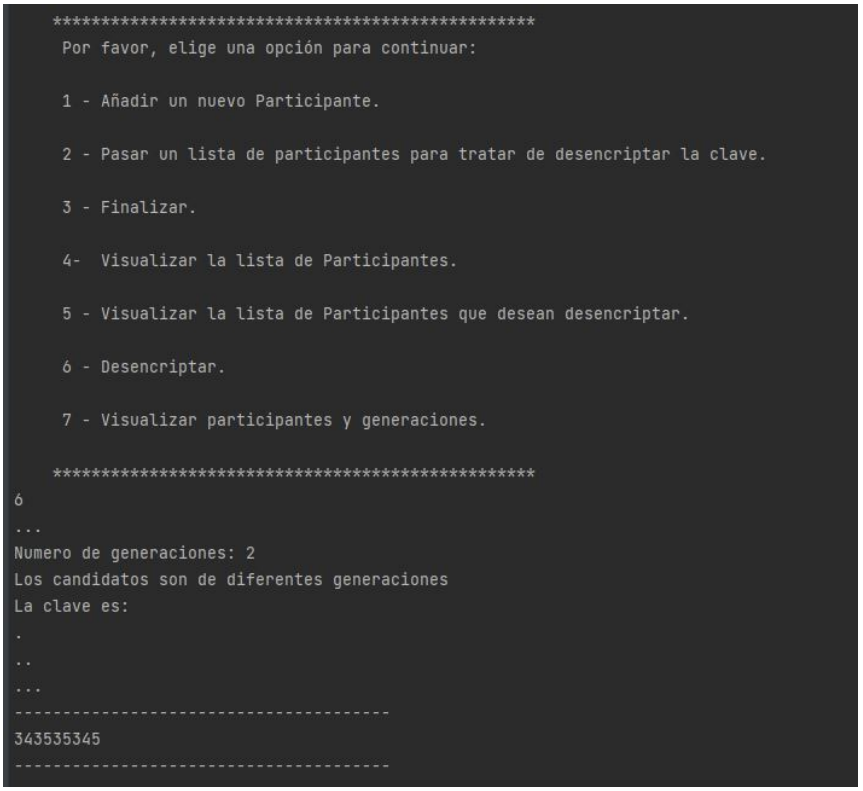
Vamos introduciendo participantes con la opción 1, escribiendo sus nombres hasta tener una lista:



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



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



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, …, P_k-1}
g1 = {P_k, …, P_k^2-1}
…

Se define ***A*** como el conjunto

A = (c_0, c_1, …, c_i) , donde

c_0 seria el numero participantes de *g0* que se postulan para relevar el secreto,
c_1 seria 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)

$$c\_0 + c\_1 + \ldots < k$$

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

S_A = SprevA + r (* la suma es suma directa)

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

SSS(key = r, k = c_0, n = size(g0)) para la generación g0,
SSS(key = r, k = c_1, n = size(g1)) para la generación g1,

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,

$$c\_0 + c\_1 + \ldots = k \text{ (threshold)}$$

Entonces tomamos

S_A, que será igual a *S + r_1 + r_2 +.....* (Suma directa)

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

S*SS(key = S_A, k = c_i, n = size(g_i)) con i=0, 1, 2, …* 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 +....*
   - *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 + … + c_i = k*

   c_0 miembros de g0 recuperarán r_0
   c_1 miembros de g1 recuperarán r_1
   …
   c_i miembros de g1 recuperarán S + r_0 + r_1 + …

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

[3] https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

[4] https://github.com/ManuelJoseMora/SecretSharingInfinitly