Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA (SG)

Grado en Ingeniería Informática de Servicios y Aplicaciones

COMPARTICIÓN DE SECRETOS CON USUARIOS TRAMPOSOS

Alonso García Bermejo

José Ignacio Farrán Martín

Índice

1. Introducción

Hoy en día, con el uso de las nuevas tecnologías, se comparten muchos datos por Internet. Es por eso que algunas personas intentan adentrarse en las comunicaciones para poder hacerse con datos personales, como por ejemplo DNIs, cuentas bancarias o contraseñas, entre otros.

Esto hace que sea necesaria la creación de protocolos seguros para que estas personas les sea más difícil hacerse con estos datos valiosos para las personas y que no sean usados en su contra.

El problema surge cuando dentro de estos protocolos seguros, las personas siguen creando métodos con los que se puede romper la seguridad de estas comunicaciones, convirtiéndose así en luchas para los desarrolladores por crear comunicaciones seguras y los llamados "tramposos", que siguen buscando las formas de acceder a los secretos que se comparten dentro.

Existe un concepto llamado **Compartición de secretos** en el que, como su nombre indica, consiste en la división de un secreto en partes que se reparten entre varios participantes bajo ciertas condiciones:

- Existen varios conjuntos de participantes puede reconstruir el secreto original(Suelen usarse los esquemas umbrales en los que cualquier conjunto de participantes con un número de elementos superior al umbral previamente establecido son autorizados).
- El otro conjunto de participantes no tiene permitido acceder a la información sobre el secreto.

Estas partes del secreto son obtenidas a partir de ciertos algoritmos, también necesario para su recuperación. Una vez obtenidas las salidas del algoritmo de "fragmentación", se utilizará otro algoritmo diferente para obtener el secreto original.

Estos mecanismos son usados en casos que requieran de una información previamente tratada a alto nivel, como la compartición de una clave de acti-

vación de bomba atómica entre varios países aliados o para la realización de transferencias y extracciones de dinero de una cuenta con varios poseedores.

Es por este motivo por lo que necesitamos que este tipo de esquemas tengan un cierto grado de seguridad para evitar el descubrimiento de los secretos.

La forma de hacer trampa en este tipo de esquemas es conseguir que un participante ofrezca un fragmento incorrecto en el momento de la reconstrucción del secreto y así, poder reconstruir el secreto con el resto partes que han sido ofrecidas e impidiendo que los otros integrantes no obtengan el secreto correcto.[20]

Los esquemas de compartición de secretos fueron introducidos por Adí Shamir y George Blakley en 1979, siendo dos propuestas diferentes tenían que compartían un mismo objetivo: Encontrar un método de para la compartición de secretos. A partir de ellos, podemos extraer una característica muy importante y es que dado un grupo de n participantes, solamente las agrupaciones de t participantes fijados con anterioridad podrían recuperar el secreto. Gracias a esto,0 posteriormente fueron creados los esquemas que permitirían que cualquier agrupación autorizada puede realizar un esquema de compartición de secretos.[2]

1.1. Objetivos

En este proyecto se hará una investigación para hacer el protocolo de Shamir un medio más seguro para la compartición de secretos. Para ello, tendremos en cuenta los siguientes objetivos:

- **Objetivo 1:** Estudio teórico del protocolo de Shamir para la compartición de secretos.
- **Objetivo 2:** Estudio teórico de los cuerpos de Galois.
- **Objetivo 3:** Búsqueda bibliográfica para entender propuestas existentes en la literatura sobre la prevención de trampas.
- **Objetivo 4:** Estudio teórico de varias posibles soluciones a la existencia de trampas.
- **Objetivo 5:** Implementación y experimentación de las soluciones propuestas en Python.
- **Objetivo 6:** Análisis de los resultados obtenidos en la experimentación.

2. Planificación

En esta sección veremos la planificación tomada en el proyecto, la estimación del esfuerzo necesario para completar el proyecto, la creación de presupuestos tanto de hardware como de software y los recursos humanos necesarios para la investigación e implementación del proyecto.

2.1. Estimación de esfuerzo

Comenzaremos con una estimación del esfuerzo (tiempo que se requiere para el desarrollo del proyecto). Para ello, utilizaremos las técnicas del diagrama de PERT para estimar el tiempo.

El método consiste en escoger la ruta crítica del proyecto (la secuencia más larga de tareas que se deben realizar para la finalización del proyecto), de la cual usar la fórmula de PERT para calcular el tiempo, teniendo en cuenta que las tareas se pueden estimar de formas diferentes:

$$T = \frac{T_{Optimista} + 4T_{Probable} + T_{Pesimista}}{6} \tag{1}$$

- $T_{Optimista}$ La mínima cantidad de tiempo para completar una tarea.
- $T_{Probable}$ La cantidad de tiempo más probable en la que terminará la tarea.
- $T_{Optimista}$ La máxima cantidad de tiempo para completar una tarea.

Como este es un proyecto realizado por una sola persona, solo existe una ruta, por lo que esa será la ruta crítica. Esto ayudará a construir el diagrama de Gantt más adelante usando los tiempos estimados en este apartado.

A continuación, veremos una tabla con todas las tareas y su resultado calculado, donde la unidad de tiempo estándar serán las horas.

Estimación				
Tarea	**Tiempo optimista**	**Tiempo probable**	**Tiempo pesimista**	**Tiempo estimado**
Reunión de inicio	1	1	2	1
Estimación de esfuerzo	10	15	20	15
Cálculo de presupuestos	6	8	10	8
Asignación de tareas	3	5	8	5.16
Investigación sobre el funcionamiento de Shamir	9	12	15	12
Investigación sobre como mejorar Shamir	12	16	20	16
Redacción sobre las investigaciones	15	17	20	17.1
Instalación de programas necesarios	1	2	3	2
Estudio de la arquitectura del programa	7	9	15	9.6
Creación de diagramas de flujo	15	17	20	17.16
Investigación sobre que librerías usar para Shamir	2	4	6	4
Implementación de Shamir en Python	25	32	45	25.83
Pruebas y corrección de Shamir	12	14	16	14
Redacción sobre la arquitectura y diagramas	10	15	20	15
Redacción sobre la implementación de Shamir	17	20	25	20.33
Investigación sobre qué librerías usar para las mejoras	1	3	5	3
Implementación de las mejoras	25	32	40	32.16
Pruebas y corrección de las mejoras	6	10	15	10.1
Redacción sobre la implementación	15	18	22	24.1
Redacción sobre las librerías usadas	10	15	20	15
Redacción de pruebas	15	17	20	17.16
Corrección de la memoria	15	20	30	20.83

Tiempo total
304.53

Tabla 1: Estimación de tiempo

2.2. Roles

Los roles de un proyecto son los encargados de gestionar las fases de un proyecto. Esto nos servirá para poder hacer el cálculo de presupuestos más adelante.

Como ya hemos mencionado anteriormente, estamos tratando un proyecto de investigación desarrollado por un miembro de trabajo, por lo que el único rol que estará presente es el de **investigador**, que será el encargado de realizar todas las tareas del proyecto.

También se podrían considerar otros roles como **programador**, **jefe de proyecto** o **analista**, pero al no ser un desarrollo de software quedan descartados.

2.3. Diagrama de Gantt

Su función es representar las tareas de un proyecto de forma visual y ordenadas cronológicamente, pudiendo ver así la duración en meses del proyecto que se usará para el cálculo de los presupuestos.

Con el proyecto terminado, observamos que la duración del proyecto han sido 7 meses y 5 días (**321 horas totales**), teniendo en cuenta que los días festivos y fines de semana no se trabaja y una media de **3 a 4 horas** de trabajo diarias.

	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1		Reunion de inicio	1 día	vie 30/06/23	vie 30/06/23		Investigador
2		Estimación de esfuerzo	5 días	lun 03/07/23	vie 07/07/23	1	Investigador
3		Calculo de presupuestos	3 días	lun 10/07/23	mié 12/07/23	2	Investigador
4		Asignación de tareas	2 días	jue 13/07/23	vie 14/07/23	3	Investigador
5		Investigación sobre el funcionamiento de Shamir	7 días	lun 17/07/23	mar 25/07/23	4	Investigador
6		Investigación sobre como mejorar Shamir	8 días	mié 26/07/23	vie 04/08/23	5	Investigador
7		Redacción sobre las investigaciones	3 días	lun 07/08/23	mié 09/08/23	6	Investigador
8		Instalación de programas necesarios	2 días	jue 10/08/23	vie 11/08/23	7	Investigador
9		Estudio de la arquitectura del programa	4 días	lun 14/08/23	jue 17/08/23	8	Investigador
10		Creación de diagramas de flujo	5 días	vie 18/08/23	jue 24/08/23	9	Investigador
11		Investigación sobre que librerias usar para Shamir	4 días	vie 25/08/23	mié 30/08/23	10	Investigador
13		Pruebas y corrección de Shamir	4 días	jue 28/09/23	mar 03/10/23	11	Investigador
14		Redacción sobre la arquitectura y diagramas	4 días	mié 04/10/23	lun 09/10/23	13	Investigador
15		Redacción sobre la implementacion de Shamir	3 días	mar 10/10/23	jue 12/10/23	14	Investigador
16		Investigación sobre que librerias usar para las mejoras	3 días	vie 13/10/23	mar 17/10/23	15	Investigador
17		Implementación de las mejoras	26 días	mié 18/10/23	mié 22/11/23	16	Investigador
18		Pruebas y corrección de las mejoras	7 días	jue 23/11/23	vie 01/12/23	17	Investigador
20		Redacción sobre las librerias usadas	3 días	jue 14/12/23	lun 18/12/23	18	Investigador
21		Redacción de pruebas	4 días	mar 19/12/23	vie 22/12/23	20	Investigador
22		Corrección de la memoria	20 días	mar 09/01/24	dom 04/02/24	21	Investigador

Figura 1: Asignación de tareas

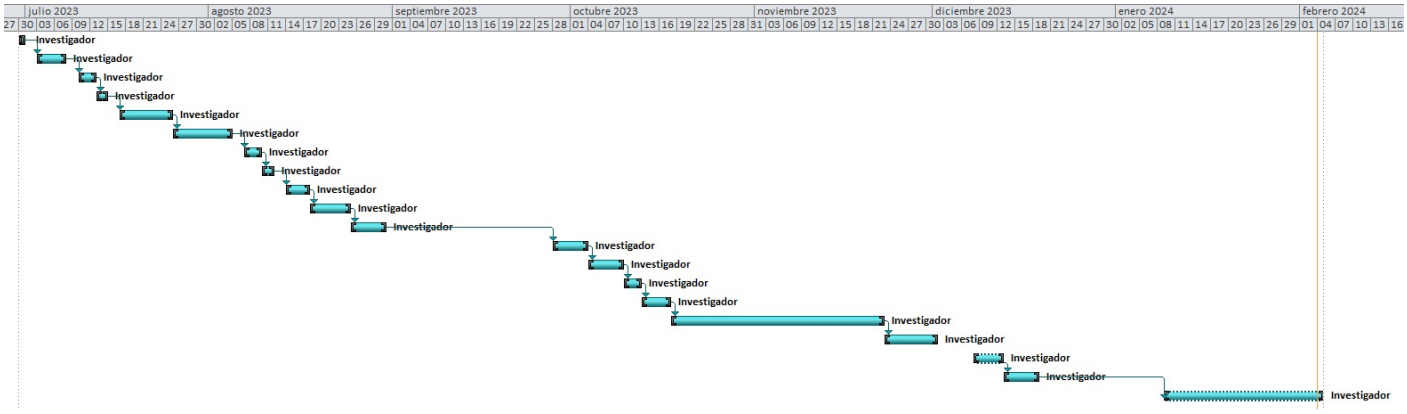

Figura 2: Diagrama de Gantt

2.4. Presupuestos

2.4.1. Software

Aquí se verá el costo de los programas utilizados durante el tiempo estimado. Como el periodo es de 7 meses y 5 días, se estimará sobre 7 meses a causa de que algunos programas funcionan con suscripciones mensuales.

Para calcular el costo de las licencias se aplicará un porcentaje de descuento del precio total proporcional a la dedicación del proyecto.

Presupuesto de software			
Programa	**Costo por mes**	**Uso proporcional**	**Costo total**
Windows 11	148€	10 %	14.80€
Visual Studio Code	Gratuito	40 %	0€
LaTeX	Gratuito	60 %	0€
StarUML	117.80€	20 %	23.56€
Overleaf	19€	60 %	79.80€
Microsoft Project	11.37€	10 %	7.96€
GitHub	Gratuito	10 %	0€
Pyhton (Y sus librerías)	Gratuito	30 %	0€

Tabla 2: Presupuesto software

Obtenemos un total de **126.12€**.

2.4.2. Hardware

En este apartado se incluirán todos los medios "físicos" para la realización del proyecto. Se verán dos tablas en las que se incluya un equipo apto para la investigación y otra con otros componentes necesarios.

El equipo montado se basará en los requisitos **mínimos** que pide el software necesario para la investigación.

Al igual que con el apartado de software, también se le aplicará un descuento proporcional, que en este caso será igual para todos los componentes y será de un 20 %.

El precio mostrado será con el descuento ya aplicado:

Componentes de equipo recomendados		
Componente	**Modelo**	**Precio**
Procesador	AMD Ryzen 3 3200G 3.6 GHz BOX	17.40€
Placa base	ASRock B450M Pro4 R2.0	16€
RAM	Kingston FURY Beast DDR4 2666 MHz 4GB CL16	5€
Torre	Tacens ALUX USB 3.0 Negro	5.6€
Fuente de alimentación	Tacens Anima APIII500 500W	3.40€
Disco duro	Kingston A400 SSD 120GB	3.35€
Monitor	Asus VA249HE 23.8"LED FullHD	17€
Teclado y ratón	Trust Taro Pack Teclado y Ratón	3.2€

Tabla 3: Presupuesto de equipo

Presupuesto de hardware		
Hardware	**Uso proporcional**	**Precio**
Acceso a internet (7 meses)	60 %	96.18€
Servidor externo	70 %	468.3€

Tabla 4: Presupuesto de hardware

En total: 635.43€

2.4.3. Presupuestos de roles

Como se ha mencionado antes, el único rol en este proyecto será el investigador, así que el presupuesto total de esta sección será el sueldo de un investigador (preferiblemente cercano a los temas a tratar) durante los meses en el que el proyecto está en progreso.

Lo más parecido que hay a este puesto es el **analista de ciberseguridad,** cuyo sueldo aproximado es de 2700€ al mes (teniendo una jornada laboral completa de 8 horas diarias y 40 horas semanales). Conociendo estos datos obtenemos que ganará aproximadamente **16.87€** por hora.

El proyecto en total ha tardado en realizarse en aproximadamente **321 horas**. Lo que implicará que el investigador cobrará por todo el proyecto **5416.87€**.

2.4.4. Presupuesto total

Una vez calculados todos los presupuestos por separado, el presupuesto total de este proyecto es de **6718.42€**. Debemos tener en cuenta de que esto es un presupuesto aproximado que puede variar en el desarrollo del proyecto.

3. Preliminares teóricos

3.1. Criptografía

La criptografía es un proceso que consiste en proteger la información valiosa mediante algoritmos y convertirla en algo inteligible, pudiendo ser únicamente descifrado por la persona a la que le debe de llegar estos datos.[15]

Mediante esta práctica conseguimos crear comunicaciones seguras en presencia de terceras personas, cuyos objetivos son:

- **Confidencialidad**: Los datos solo pueden ser vistos por las personas autorizadas
- **Integridad**: Los datos no deben de ser alterados en el tránsito.
- **Autenticación**: Se asegura que el emisor y el receptor son la persona que dicen ser.
- **Sin rechazo**: Evita que un usuario de la comunicación rechace los compromisos de creación y transmisión de la información[4].

3.1.1. Algoritmos criptográficos

Son los algoritmos utilizados para la transformación de datos "legibles" en datos "ilegibles" y viceversa. Se pueden distinguir dos tipos:

- **Algoritmos de clave simétrica:** Se utiliza la misma clave para cifrar y descifrar el mensaje. El problema de este tipo de algoritmos es que esa clave debe de ser enviada de forma segura al receptor para que pueda descifrar el mensaje.

  El proceso consiste en que el emisor cifrará su mensaje usando la clave secreta. Después de esto, se enviará el mensaje y el receptor, con la misma clave, descifrará el mensaje.

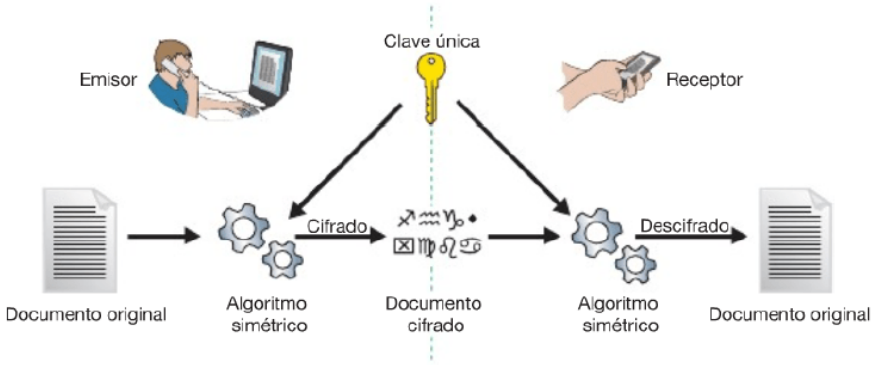

Figura 3: Clave simétrica

- **Algoritmos de clave asimétrica:** En este caso se utilizan 2 claves para la transmisión del mensaje:

  - **Clave pública**: Es la clave que cualquier usuario podrá conocer y sirve para cifrar el mensaje.
  - **Clave privada**: Esta será la clave que solo el propietario podrá conocer y no compartirá con nadie. Sirve para descifrar el mensaje.[6]

  El inconveniente de este proceso es que la velocidad de la operación es mucho más lenta que la de la clave simétrica, pero, a la vez es más segura, ya que no es necesario la compartición de una clave entre los usuarios.

  El proceso consiste en que el emisor usará la clave pública del receptor para cifrar el mensaje y lo enviará al receptor. Una vez recibido, usará su clave privada para descifrar el mensaje[13].

Figura 4: Clave asimétrica

3.1.2. HASH

Son funciones matemáticas cuya función es transformar una serie de datos en una cadena de caracteres de longitud fija y única para los datos introducidos. También es importante qué sea irreversible, es decir, que sea prácticamente imposible obtener la cadena original a partir de Hash.[18]

Existen muchos tipos de Hash y cada uno con sus variantes, como MD5 (evolución de MD2, MD3...) y SHA (SHA1, SHA512 o SHA224).

Estas funciones se usan principalmente en:

- **Recuperación de datos**: al conjunto de datos se le asigna un valor. Esto hará que la búsqueda se restrinja y se pueda detectar la ubicación de los datos de forma mucho más sencilla.[17]
- **Detección de malware**
- **Gestión de contraseñas**
- **Firma electrónica**

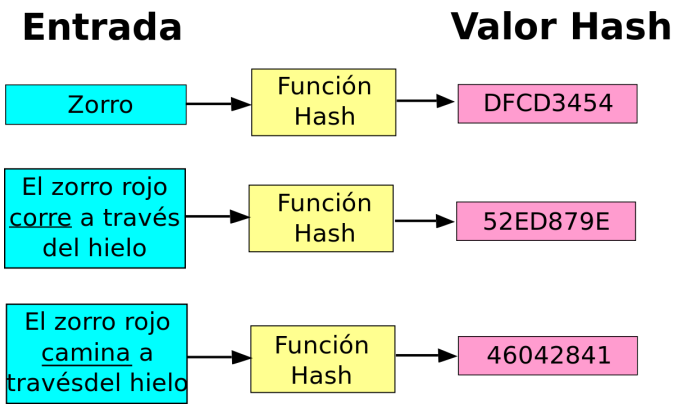

Figura 5: Ejemplo de Hash

El proyecto necesita un Hash poco colisionable para reducir las posibilidades de que el tramposo pueda modificar sus puntos para que siga dando el mismo hash del secreto real.

3.2. Protocolo de Shamir

Es un algoritmo criptográfico creado para la compartición de secretos por Adi Shamir en 1979. Su funcionamiento se basa en el método de interpolación de Lagrange.

Consiste en que el secreto a compartir se divide en partes asignándose cada una a un usuario.

También se le conoce como **protocolo sin clave de Shamir**, ya que en esta comunicación no se intercambian las claves entre emisor y receptor.[10]

3.2.1. Conceptos matemáticos

El objetivo es dividir el secreto en un conjunto de datos D en **n** partes $\{D_1, ...D_n\}$ teniendo en cuenta:

- La existencia de k o más D_i hace que sea fácilmente computable.
- El conocimiento de $k-1$ o menos D_i hace que D sea indeterminado, es decir, que sea imposible obtener le secreto.

A esto se le llama umbral, lo que significa que sí $k = n$ se necesitaran a todos los usuarios para reconstruir el secreto[7].

3.2.2. Funcionamiento

La idea principal de Shamir es la creación de polinomios, siendo necesarios $n+1$ puntos para la creación de un polinomio de grado n, si quisiéramos compartir un secreto S, seleccionando $k-1$ coeficientes a_n (siendo $k < n$) teniendo en cuenta que el coeficiente $a_0 = S$ y construimos el polinomio: $f(x) = a_{k-1}x^{k-1} + ... + a_2x^2 + a_1x + a_0$

Se calcularán n puntos utilizando $x = 1, x_2 = 2...x_n = n$ y aplicando $f(x)$ obtendremos los puntos $(x, f(x))$ de los que se deberán escoger k puntos. Al aplicar interpolación de Lagrange[7] se consigue un polinomio $P(x)$ cuyo $P(0) = S$.

3.2.3. Ejemplo práctico

Tenemos un secreto $S = 12$ en una red de $n = 5$ usuarios, y queremos utilizar cualquier subconjunto $k = 3$ para que el secreto sea reconstruido.

Elegimos $k-1$ números: $a_1 = 34$, $a_2 = 199$.

Así, construiremos el polinomio: $f(x) = 34x^2 + 199x + 12$ y calcularemos los $f(x)$ correspondientes de cada usuario para obtener siguientes puntos: $\mathbf{(1, 245); (2, 546); (3, 915); (4, 1352); (5, 1857)}$.

Una vez obtenidos los puntos, seleccionaremos un subconjunto $\boldsymbol{k = 3}$: $\mathbf{(2, 546); (3, 915); (5, 1857)}$ y utilizaremos la interpolación de Lagrange:

- $\boldsymbol{l_0 = \frac{(x-x_1)}{(x_0-x_1)} \cdot \frac{(x-x_2)}{(x_0-x_2)} = \frac{(x-3)}{(2-3)} \cdot \frac{(x-5)}{(2-5)} = -\frac{(x-3)}{1} \cdot -\frac{(x-5)}{3}}$
- $\boldsymbol{l_1 = \frac{(x-x_0)}{(x_1-x_0)} \cdot \frac{(x-x_2)}{(x_1-x_2)} = \frac{(x-2)}{(3-2)} \cdot \frac{(x-5)}{(3-5)} = \frac{(x-2)}{1} \cdot -\frac{(x-5)}{2}}$
- $\boldsymbol{l_2 = \frac{(x-x_0)}{(x_2-x_0)} \cdot \frac{(x-x_1)}{(x_2-x_1)} = \frac{(x-2)}{(5-2)} \cdot \frac{(x-3)}{(5-3)} = \frac{(x-2)}{3} \cdot \frac{(x-3)}{2}}$

Aplicando:

$$\sum_{j=1}^{k} \boldsymbol{y_j \cdot l_j(x)}$$

$\boldsymbol{f(x) = 546 \cdot (-\frac{(x-3)}{1} \cdot -\frac{(x-5)}{3}) + 915 \cdot (\frac{(x-2)}{1} \cdot -\frac{(x-5)}{2}) + 1857 \cdot (\frac{(x-2)}{3} \cdot \frac{(x-3)}{2})}$

El mensaje será el resultado de: $\boldsymbol{f(0) = 12}$

3.3. Cuerpos de Galois

También conocidos como cuerpos finitos, son una estructura algebraica compuesta por un número finito de elementos que fueron estudiados por Évariste Galois en 1830. Están determinados por su cardinal, es decir, un número primo elevado a una potencia. Los cuerpos finitos se usan en muchos campos de las matemáticas como teoría de números, geometría algebraica o criptografía.

El cuerpo finito del cardinal $\boldsymbol{q}$ se representa como $\boldsymbol{F_q}$ o $\boldsymbol{GF(q)}$.

Utilizaremos esta estructura para tener un número de bits fijo. Esto servirá para evitar los problemas de redondeo en el caso de dar como resultado un número en coma flotante.

3.3.1. Construcción de los cuerpos finitos

La construcción de un cuerpo finito $\boldsymbol{F_q}$ con $\boldsymbol{q = p^n}$ es similar a la de los números complejos a partir de los números reales.

- 1º- Se busca un polinomio irreducible $\boldsymbol{m(\alpha)}$ de grado $\boldsymbol{r}$ con coeficientes en cuerpo primo $\boldsymbol{F_q = Z_p}$.
- 2º- Se consideran todos los posibles polinomios en la variable $\boldsymbol{\alpha}$ y se reducen a $\boldsymbol{m(\alpha)}$, es decir, solo los restos de la división por $\boldsymbol{m(\alpha)}$.
- 3º- Los posibles restos son de grado $\leq \boldsymbol{r-1}$ y el número de ellos es $\boldsymbol{p^r}$ (hay $\boldsymbol{p}$ posible valores para los coeficientes de grado 0,1, ..., $\boldsymbol{r-1}$)

El cuerpo que utilizaremos será $\boldsymbol{F_q = 2^{64}}$ cuyo polinomio primitivo es: $\boldsymbol{x^{64} + x^{33} + x^{30} + x^{26} + x^{25} + x^{24} + x^{23} + x^{22} + x^{21} + x^{20} + x^{18} + x^{13} + x^{12} + x^{11} + x^{10} + x^7 + x^5 + x^4 + x^2 + x + 1}$

3.3.2. Propiedades de los cuerpos de Galois

- Todos los elementos de $\boldsymbol{F_q}$ cumplen la siguiente ecuación: $\boldsymbol{x^q - x = 0}$.

Demostración: Dado un campo finito $\boldsymbol{F_q}$ de orden $\boldsymbol{q = p^n}$ (siendo $\boldsymbol{p}$ un número primo) el grupo multiplicativo $\boldsymbol{F_q^*}$ es de orden $\boldsymbol{q-1}$, por lo que para todo $\boldsymbol{x \in F_q^* : x^{q-1}}$.
Implica que: $\boldsymbol{x^q - x = 0}$

- Dado un cuerpo, F_q su grupo multiplicativo F_q^* es un grupo cíclico de orden $q-1$. Esto significa que siempre hay al menos un elemento $x \in F$ tal que $F = \{0, 1, x, x^2, ..., x^{q-2}\}$ esos elementos x reciben el nombre de **elementos primitivos**.

Demostración: Sea G uno de los subgrupos de F_q^*, de orden de $p_1^{e_1}, p_2^{e_2}, ..., p_k^{e_k}$, por el teorema fundamental de los grupos abelianos $G \cong Z_{p_1^{e_1}} \otimes Z_{p_2^{e_2}} \otimes ... \otimes Z_{p_k^{e_k}}$.
Sea m el m.c.m dé $p_1^{e_1}, p_2^{e_2}, ..., p_k^{e_k}$.
Dado que todo $x \in G$ cumple $x^r - 1 = 0$ para un r que divide a m, entonces se cumple $x^m - 1 = 0$, teniendo como máximo m raíces en F.
Entonces: $n \leqslant m$.
Por otro lado, $m \leqslant |G|$ que implica qué $m = n$.
Por lo tanto, G contiene un elemento de orden, n siendo así cíclico.

- El cuerpo de F_q ($q = p^m$) contiene una $F_{q\prime}$ ($q\prime = p^n$) si y solo si n divide a m. En esta situación $F_{q\prime}$, es un subcuerpo de F_q y F_q es una extensión dé $F_{q\prime}$.

Demostración: Es inmediato que $[F_q : F_p] = m$ y que $[F_{q\prime} : F_p] = n$.
No es posible que $F_P \subset F_{q\prime} \subset F_q$ a no ser que n divida a m.
Suponiendo que $m = n$ y sustituyendo $y = p^n$ en la ecuación $y^m - 1 = (y-1) \cdot (y^{m-1} + ... + y + 1)$, se obtiene que $q\prime - 1$ divide $q - 1$.
Puesto que el grupo multiplicativo F_q^x es cíclico de orden $q-1$ y $q\prime - 1$ divide a $q - 1$.
Existe $\beta \in F_q^x : \beta^{q\prime - 1} = 1$ cuyas potencias $q\prime - 1$ satisfacen $x^{q\prime - 1} - 1 = 0$.
Por lo tanto, $x^{q\prime} - x$ descompone completamente a $F_{q\prime}$y sus raíces forman un cuerpo de orden $q\prime$.

4. Metodología

Como metodología de trabajo se ha optado por RUP o Proceso Unificado de Desarrollo de Software, debido a que al no ser una aplicación software como tal se ha considerado este marco de trabajo por su flexibilidad.

En esta sección se verán sus fases, además de cuál será proceso que se seguirá para la elaboración del código y las herramientas utilizadas para su elaboración.

4.1. Fases del proceso

Dentro del Proceso Unificado de Desarrollo de Software existe un ciclo de desarrollo que constituye la vida del sistema, donde se pueden observar 4 fases importantes: **Inicio**, **Elaboración**, **Construcción** y **Transición**.

Una vez acabado la primera iteración del ciclo, este se reiniciará y volverá a comenzar para añadir nuevas mejoras y funcionalidades al producto.

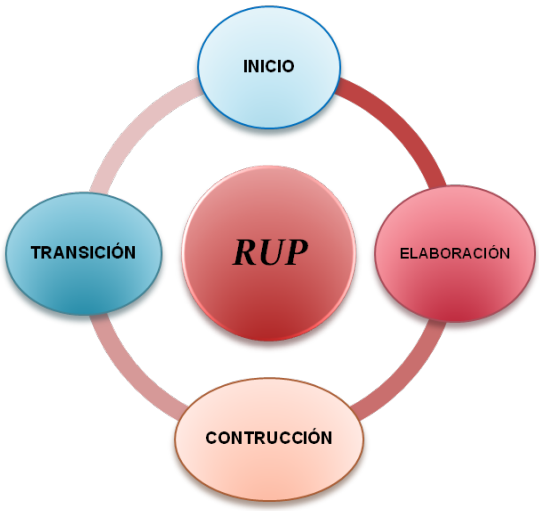

Figura 6: Ciclo de desarrollo del RUP

- **Inicio**: En esta primera etapa se centra en la definición del proyecto, su alcance y preparación. Es aquí donde se estudia qué es y cómo funciona el protocolo Shamir, además de la búsqueda de información de qué y cómo se pueden implementar las mejoras para hacerlo más seguro.
- **Elaboración**: Se estudia la forma en la que se implementará el código, qué lenguaje se utilizará, que librerías y como se estructurará, además de empezar a documentar su elaboración y qué técnicas se utilizarán para alcanzar los resultados deseados.
- **Construcción**: Esta es la etapa más puramente práctica. En ella, se empieza la construcción del código y de las mejoras ya planteadas en la anterior fase. También se hacen las pruebas necesarias para documentar y comprobar el correcto funcionamiento de software.
- **Transición**: Se corrigen los errores detectados en la etapa anterior y se plantean nuevas mejoras o sugerencias para mejorar la calidad del producto.

En este caso concreto, el producto a desarrollar está en constante evolución, por lo que se considera que siempre puede haber más margen de mejora, y que el proceso de desarrollo puede llegar a alargarse en el tiempo hasta que se considere que el margen de mejora de una versión a otra es mínimo.[19]

4.2. Elaboración del código

El protocolo será programado con el lenguaje Python, y con esa base se irán implementando las diferentes propuestas, además de un menú con el que poder ir haciendo pruebas.

Para la programación de Shamir, es necesario la utilización de cuerpos finitos para la recuperación del mensaje, por lo que se requiere de una librería específica llamada Galois, que nos permite crear campos de cuerpos finitos con las características que especifiquemos. En este caso, usaremos un campo de cuerpos de orden $\mathbf{2^{64}}$, dado que no usamos números reales. En vez de la interpolación de Lagrange deberemos utilizar un sistema de ecuaciones para

conseguir la ecuación: Para ello, necesitaremos crear una matriz de Vandermonde que consiste en:

$$\begin{bmatrix} \mathbf{1} & \boldsymbol{x_1} & \boldsymbol{x_1^2} & \boldsymbol{...} & \boldsymbol{x_1^{n-1}} \\ \mathbf{1} & \boldsymbol{x_2} & \boldsymbol{x_2^2} & \boldsymbol{...} & \boldsymbol{x_2^{n-1}} \\ \mathbf{1} & \boldsymbol{x_3} & \boldsymbol{x_3^2} & \boldsymbol{...} & \boldsymbol{x_3^{n-1}} \\ \mathbf{1} & \boldsymbol{x_n} & \boldsymbol{x_n^2} & \boldsymbol{...} & \boldsymbol{x_n^{n-1}} \end{bmatrix} \tag{2}$$

Siendo $\boldsymbol{x_n}$ los $\boldsymbol{x_i}$ de los puntos seleccionados, para obtener la ecuación necesaria y asi recuperar el mensaje, se deberá multiplicar a la matriz de Vandermonde por los $\boldsymbol{f(x)}$:

$$\begin{bmatrix} \boldsymbol{f(x_1)} \\ \boldsymbol{f(x_2)} \\ \boldsymbol{f(x_3)} \\ \boldsymbol{f(x_n)} \end{bmatrix} \tag{3}$$

De esta manera se obtendrá la ecuación y solo tendríamos que aplicar $\boldsymbol{f(0)}$ a la misma para recuperar el secreto.

En esta simulación, los $\boldsymbol{k}$ puntos serán seleccionados aleatoriamente para agilizar las pruebas más adelante.

4.3. Herramientas utilizadas

Durante el proceso de desarrollo del proyecto se utilizarán varias herramientas:

- **Windows 11:** Es el sistema operativo en el que se ha trabajado con todas las aplicaciones.
- **Visual Studio Code:** Es un editor de código totalmente personalizable con la oportunidad de la instalación de múltiples extensiones como estilizadores de código o herramientas para resolución de fallos.
- **LaTeX:** Es un procesador de texto diseñado para la creación de documentos con la posibilidad de la generación de expresiones matemáticas de manera más sencilla.

- **StarUML:** Herramienta usada para la creación de diagramas UML.
- **Overleaf:** Herramienta online para la creación de documentos usando LaTeX.
- **Microsoft Project:** Herramienta destinada a la administración de proyectos. Será utilizada principalmente para crear el diagrama de Gantt correspondiente al proyecto.
- **GitHub:** Aplicación para el control de versiones.
- **Python:** Lenguaje de programación de alto nivel en el que se programará todo el código del protocolo, además de ciertas librerías que permiten su programación con más facilidad:
  - **random:** Genera números aleatorios. Usada generalmente para sacar varios números aleatorios dentro de un rango específico.[12]
  - **NumPy:** Encargada del cálculo numérico para volumenes de números grandes, además de otras funciones como cálculo de matrices, uso de la lógica y muchas más.[1]
  - **itertools:** Sirve para crear iteraciones para bucles. En este caso se utilizará *combinations* para crear combinaciones sin repetición.[8]
  - **Galois:** Para la creación de campos de Galois. Necesarios al hacer uso de números de grandes dimensiones.[14]
  - **hashlib:** Usada para calcular Hashes de las cadenas de texto indicadas.

El lenguaje de programación y las librerías se explicarán más detalladamente en el apartado de "Lenguaje de programación y librerías".

5. Lenguaje de programación y librerías

5.1. Python

Es un lenguaje de programación creado a finales de los años 80 como sucesor del ABC. Utilizado principalmente para crear aplicaciones web[11], desarrollo de software y machine learning, además de contar con un gran volumen de librerías y una sintaxis parecida al inglés, por lo que facilita el trabajo de los desarrolladores consiguiendo que se escriban códigos con muchas menos líneas frente a otros lenguajes.[5]

Sus características más destacables son:

- **Lenguaje de programación de alto nivel**: Permite escribir expresiones similares al lenguaje habitual al de los humanos. Como ya se ha mencionado antes, presenta una sintaxis parecida al inglés. Esto hace que para que el ordenador lo entienda se deban usar compiladores que traduzcan las órdenes al lenguaje máquina.
- **Código abierto**: Es el software al que cualquiera puede acceder y ser modificado por cualquier usuario.
- **Multiparadigma**: Soporta varios paradigmas de la programación como el orientado a objetos, funcional o declarativo.
- **Interpretado**: Ejecuta sus instrucciones al momento sin la necesidad de un compilador.[3]

5.2. Librerías

5.2.1. random

Encargada de la generación de números aleatorios con una amplia selección de funciones, de la cual usaremos la generación de un número en un rango seleccionado para la selección de usuarios, que serán los encargados de hacer los cálculos de las partes del secreto.

Ejemplo 1: Generación y muestra en pantalla de un número aleatorio dentro de un rango desde 1 hasta 10.

```
import random
x = random.randint(1,10)
print(x)

Resultado:
5
```

5.2.2. NumPy

Es una de las librerías más básicas para el cálculo numérico, como álgebra lineal, lógica, creación y operaciones de matrices y más funciones. En nuestro caso, utilizaremos NumPy para resolver sistemas ecuaciones y obtener el valor numérico de un polinomio en un punto.

Ejemplo 2: Obtención del valor numérico de un polinomio en 2 y muestra por pantalla.

```
import numpy as np
coeficientes = [12,32,4]
resultado = np.polyval(coeficientes, 2)
print(resultado)

Resultado:
116
```

Ejemplo 3: Resolución de un sistema de ecuaciones y salida de los resultados por la pantalla.

```
import numpy as np
m = [[1,2],[4,5]]
x = [7,8]
resultado = np.linalg.solve(m,x)
print(resultado)

Resultado:
[-6.33333333 6.66666667]
```

5.2.3. itertools

Ofrece herramientas de iteraciones rápidas y eficientes en memoria, teniendo la posibilidad de combinarlas con otras. Esta librería, sin embargo, no permite hacer repeticiones, conteos, ciclos, entre otros.

A pesar de esto, lo que nos interesa son las herramientas de combinatoria con las que podremos crear combinaciones de elementos de todo tipo, permutaciones, combinaciones con y sin repetición o productos cartesianos.[9]

Para este proyecto en concreto usaremos las combinaciones para poder la detección de los mentirosos. Para ello, utilizaremos “combinations”, y así

crear todas las combinaciones de los $\boldsymbol{n}$ puntos posibles establecidos por $\boldsymbol{k+i}$.

Ejemplo 4: Obtención de las combinaciones de 3 elementos sin repetición de todos los elementos de un array

```
from itertools import combinations
lista = [1,2,3,4,5,6,7,8,9]
for combinación in combinations(lista,3):
  print(combinacion)

Resultado:
(1, 2, 3)(1, 2, 4)(1, 2, 5)(1, 2, 6)(1, 2, 7)...
```

5.2.4. galois

Es una extensión de la librería NumPy que nos permite usar los campos de Galois dada la necesidad de usar cuerpos finitos para el cálculo del secreto en el protocolo de Shamir. Para crear el campo, deberemos utilizar la función "galois.GF($\boldsymbol{p} * *\boldsymbol{m}$)"siendo $\boldsymbol{p}$ un número primo y $\boldsymbol{m}$ la potencia.

Ejemplo 5: Creación de un campo de Galois de orden $\mathbf{2^{64}}$

```
import galois
GF = galois.GF(2**64)
print(GF(3))

Resultado:
3
```

En el ejemplo se imprime por pantalla GF(3). Esto quiere decir que se imprimirá el número 3 representado dentro del cuerpo de Galois previamente creado ($\mathbf{2^{64}}$).

5.2.5. hashlib

Es la librería encargada de ofrecer funciones hash como SHA1, SHA256, SHA512 y MD5 para el cifrado de elementos en Python.

Para este proyecto, usaremos el algoritmo de SHA256 para cifrar el secreto a compartir y comparar el resultado con el hash del secreto calculado por los usuarios. De esta forma, podremos saber si el secreto es correcto o ha ocurrido algún problema en la comunicación.

Ejemplo 6: Aplicación del SHA256 a una cadena de caracteres y muestra por pantalla del resultado.

```
import hashlib
hash = hashlib.sha256(str("hola").encode()))
print(hash)
```

Resultado:
0x000001826C403CF0

6. Propuestas para la detección de tramposos

Como ya se mencionó antes, el objetivo de este trabajo es hacer del protocolo Shamir un protocolo más seguro mediante el planteamiento de distintas opciones y la presentación de sus pros y contras...

- **Aplicación de función Hash:** Se aplica una función Hash a los resultados obtenidos para comprobar que las operaciones se han calculado correctamente:

  - **Hash al resultado final:** Se aplica el Hash al mensaje antes de dividirlo en partes y se hace público el resultado del Hash. Una vez calculado el mensaje por los usuarios, se aplica la misma función Hash a ese resultado y se compara con el Hash anterior. Si coincide, es que es correcto y que nadie ha hecho trampas.

    Imaginemos que tenemos un secreto que queremos compartir a nuestros compañeros. Ese secreto será 1234 y usaremos el protocolo de Shamir para compartir ese secreto. Aplicando la propuesta y antes de que el secreto sea dividido en partes, se le aplicará la función hash SHA256, con la cual obtendremos este código: **a883dafc480d466ee04e0d6da986...**

    Una vez hecho el hash, procedemos a dividir el secreto, y cuando queramos recuperarlo, seguiremos el proceso normal del protocolo de Shamir. Una vez terminado, aplicaremos la misma función hash al resultado. Si coinciden, es que nadie ha hecho trampas.

  - **Hash a los puntos calculados:** Parecido al caso anterior, un servidor externo calcularía los puntos por sí mismo a la vez que lo hacen los usuarios y aplicaría la función Hash a esos resultados. Una vez que los usuarios terminasen de calcular los puntos, se compararían los resultados.

    Ahora imaginemos el caso planteado en el apartado anterior. Tenemos un secreto que es 1234 y queremos compartirlo usando el protocolo de Shamir. En este caso se dividirá el secreto en partes, y una vez que se quiera recuperar el secreto, se hará en dos paralelamente en dos secciones. Una la harán los usuarios por su cuenta y otra se encargará un servidor externo que, una vez terminado

el proceso, el servidor aplicará la función hash a los resultados calculados por los usuarios y a los suyos propios para finalmente compararlas y detectar errores.

La desventaja de esta opción es que se debe confiar en un servidor externo que haga los cálculos el mismo. Esto se debe a que el administrador del servidor puede estar de lado de los tramposos y modificar los resultados de hash para que coincidan con los de los tramposos. Además, esta opción solo sirve para detectar que se ha hecho trampas y no hace ninguna acción para arreglar el mensaje, solo se mandaría un mensaje acusatorio a las personas que hubieran hecho trampas[16].

- **Petición de k usuarios con varios usuarios redundantes:** En el momento de buscar los $\boldsymbol{k}$ necesarios para la reconstrucción del secreto, se pedirían $\boldsymbol{x}$ usuarios más y el cálculo de los datos se realizaría con los $\boldsymbol{k + x}$ datos:

  - Si hay un sospechoso, se piden $\boldsymbol{k + 1}$ y se hacen los cálculos con los $\boldsymbol{k + 1}$ datos. Si hubiera un tramposo, el sistema sería incompatible. Para comprobar que no hay tramposos, se realizarían los cálculos con $\boldsymbol{k}$ datos utilizando los $\boldsymbol{k+1}$ datos recogidos, es decir:

    Si tenemos uno $\boldsymbol{k = 3}$, deberemos hacer los cálculos con $\boldsymbol{k = 4}$. Suponiendo que tenemos un secreto $\boldsymbol{S = 12}$ y los puntos:
    $\boldsymbol{(1, 245); (2, 546); (3, 915); (4, 1352); (5, 1857)}$

    Los puntos obtenidos aleatoriamente serán:
    $\boldsymbol{(1, 245); (2, 546); (3, 915); (5, 1857)}$

    Si aplicamos el algoritmo, $\boldsymbol{f(0) = 12}$, por lo que nadie habría hecho trampas.

    En cambio, si hubiésemos recibido los puntos:
    $\boldsymbol{(1, 23); (2, 546); (3, 915); (4, 1352); (5, 1857)}$

    Con los puntos aleatorios:
    $\boldsymbol{(1, 23); (2, 546); (3, 915); (4, 1352)}$
    Aplicando el algoritmo, veríamos que saldría un sistema incompatible, por lo que sabríamos que alguien ha hecho trampas.

Ahora deberíamos comprobar quién es el que ha hecho las trampas. Para ello, debemos usar $\boldsymbol{k}$ valores y aplicar el algoritmo tantas veces sea necesario:

- Con $\mathbf{(1, 245); (2, 546); (3, 915)}$ el sistema es incompatible
- Con $\mathbf{(1, 245); (3, 915); (5, 1857)}$ el sistema es incompatible
- Con $\mathbf{(1, 245); (2, 546); (5, 1857)}$ el sistema es incompatible
- Con $\mathbf{(2, 546); (3, 915); (5, 1857)}$ el sistema es compatible

Se puede ver que el conjunto de los usuarios 2,3 y 5 da un sistema compatible, por lo que el usuario que no este dentro de este conjunto es el tramposo (en este caso es el usuario 1). Una vez visto qué sistema es compatible, se aplicaría el algoritmo para ver el resultado y obtendríamos $\boldsymbol{f(0) = 12}$. Ahora compararíamos el Hash del mensaje original con el del resultado obtenido para ver si es correcto.

- Si hay 2 sospechosos o más, se seguiría el mismo proceso que con un sospecho, haciendo combinaciones excluyendo a $\boldsymbol{x}$ usuarios. En este caso pueden ocurrir 2 escenarios diferentes:
  - Que los tramposos no se hayan puesto de acuerdo: En caso de que el sistema sea incompatible, se le aplicará la exclusión para los tramposos. En caso contrario, sería compatible y no habría problemas.
  - En el caso en el que los tramposos se hayan puesto de acuerdo: Al obtener un sistema compatible, habría que realizar una comprobación sobre el Hash del mensaje, y así detectar incongruencia. Por ende, aquellos que no sean correctos serán los tramposos. Este caso es el más complicado de los dos.

Este método tiene la ventaja de que no se necesita de un servidor externo, pero a la vez es más costoso computacionalmente, ya que cuanto más grande sea la $\boldsymbol{k}$, más cálculos habría que hacer.

Es importante recalcar que si el umbral está muy cerca de $\boldsymbol{n}$, no será posible detectar a los usuarios tramposos, ya que no se podrán hacer las combinaciones necesarias. Es decir, si por ejemplo tenemos $\boldsymbol{k = 3}$ y $\boldsymbol{n = 4}$ y queremos detectar a un usuario tramposo, al aplicar $\boldsymbol{t + 1}$ no podremos realizar ninguna combinación.

También este método solo funciona cuando la mayoría de los usuarios no son tramposos, ya que si más de la mitad fuesen tramposos, sería imposible encontrar un sistema compatible.

- **Revelar los puntos calculados de forma simultánea:** Esto es para que los tramposos no obtengan ninguna información cuando el servidor externo haya detectado trampas.

  Si existiesen tramposos, el servidor enviaría un mensaje a los usuarios indicando quiénes son y borrará los puntos enviados para que nadie pueda recuperar el secreto, para así volver a repetir el proceso, excluyendo a los tramposos previamente detectados.

7. Análisis y diseño

En este apartado veremos todo lo relacionado con la arquitectura y modelado del software desarrollado. Es un proceso de recogida de información, lo que implica el establecimiento de estructuras de datos, la representación del algoritmo y la arquitectura del mismo. Esto nos servirá para ver como de grande es la complejidad del proyecto antes de la implementación.

7.1. Requisitos

Describen las características y el comportamiento del sistema, y sirven para darnos una idea del producto que se va a desarrollar. Existen varios tipos de requisitos:

- **Requisitos de negocio**: Representan las especificaciones de alto nivel, es decir, el alcance y los objetivos del proyecto. Responden a la pregunta: ¿Qué se debe hacer?
- **Requisitos de usuario**: Describen las características que buscan los usuarios. Responden a la pregunta: ¿Para quién es el software?
- **Requisitos de software**: Describen el comportamiento del software bajo ciertas condiciones (**Requisitos funcionales**) y características o restricciones que debe tener (**Requisitos no funcionales**).

Dado que este proyecto no es una aplicación comercial y que no está orientada directamente a que la usen usuarios corrientes, solo analizaremos los requisitos funcionales y los no funcionales.

7.1.1. Requisitos funcionales

Id	Requisito
RF01	El sistema tendrá que calcular el hash del secreto cuando sea introducido por el usuario.
RF02	El sistema deberá tener un menú para poder hacer pruebas con distintas opciones.
RF03	Los datos y los resultados que se generan deberán mostrarse por pantalla.
RF04	El sistema deberá detectar qué usuarios están haciendo trampas y mostrarlo por pantalla.
RF05	El sistema deberá preguntar al usuario cuantos usuarios desea crear.
RF06	El sistema deberá preguntar al usuario que secreto quiere compartir.
RF07	El sistema deberá preguntar al usuario que umbral desea usar.
RF08	El sistema deberá preguntar al usuario cuantos usuarios deseas crear.
RF09	Si se inicia la simulación del protocolo y no se han introducido los datos, el sistema avisará al usuario de que deberá introducir los datos primero.
RF10	Una vez terminada la simulación, el sistema deberá calcular el hash del secreto calculado y compararlo con el hash del secreto compartido.
RF11	El sistema deberá avisar al usuario si el hash coincide con el hash del secreto calculado.

Tabla 5: Requisitos funcionales

7.1.2. Requisitos no funcionales

Id	Requisito
RNF01	El sistema debe ser compatible con cualquier sistema operativo.
RNF02	El sistema utilizará el algoritmo SHA256 para cifrar.
RNF03	El orden de los cuerpos finitos será $\mathbf{2^{64}}$

Tabla 6: Requisitos no funcionales

7.1.3. Diagrama de flujo

Representa la secuencia de pasos de un algoritmo. De esta forma podremos hacernos una idea visual de como funciona el algoritmo.

Debido a que el software presenta varias opciones, se harán diferentes diagramas de flujo para ver el funcionamiento de cada una.

- **Diagrama de flujo de Menú**: Aquí se representan las diferentes secuencias, de cada opción del menú.
  Las opciones 2, 3 y 4 se representarán más adelante cada una con su diagrama.

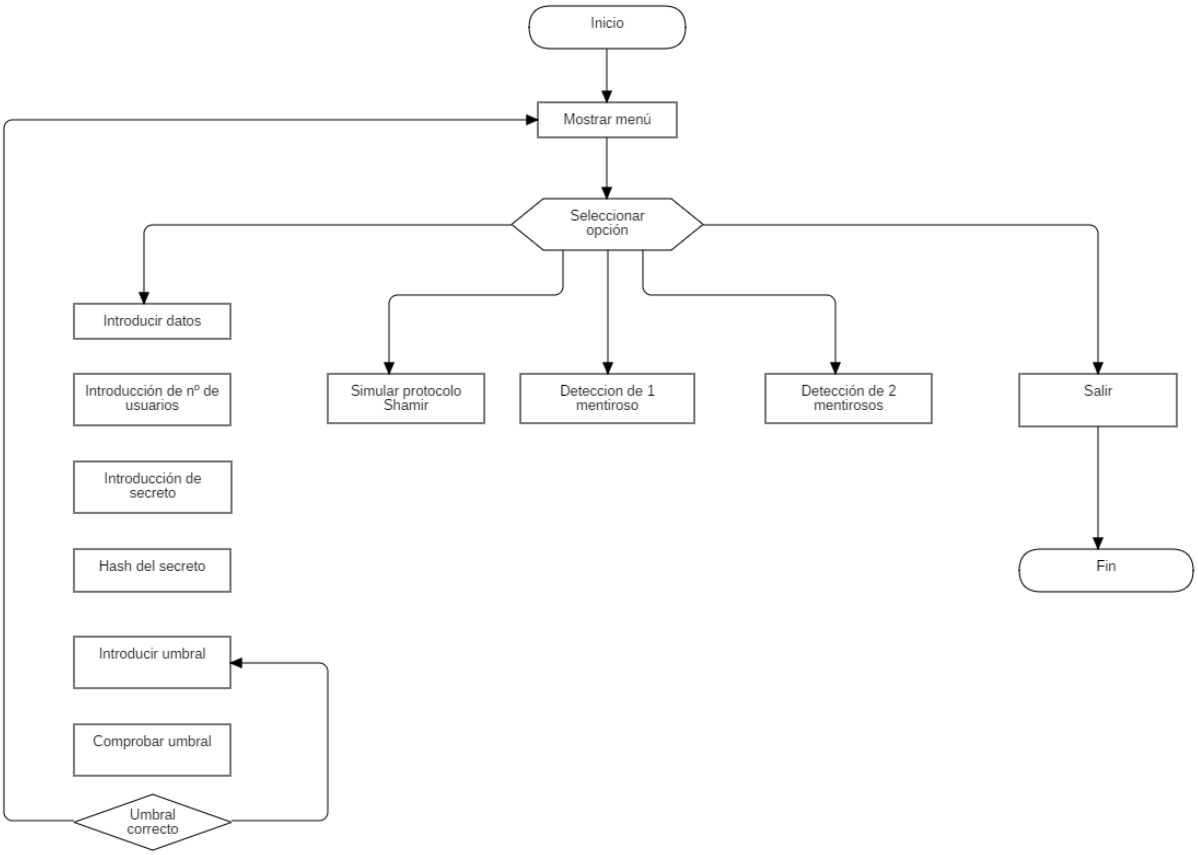

Figura 7: Diagrama de flujo Menú

- **Diagrama de flujo de Shamir (Opción 2)**: Este es el esquema principal del programa, donde se representa los pasos que se dan para simular la compartición de un secreto usando el protocolo de Shamir.

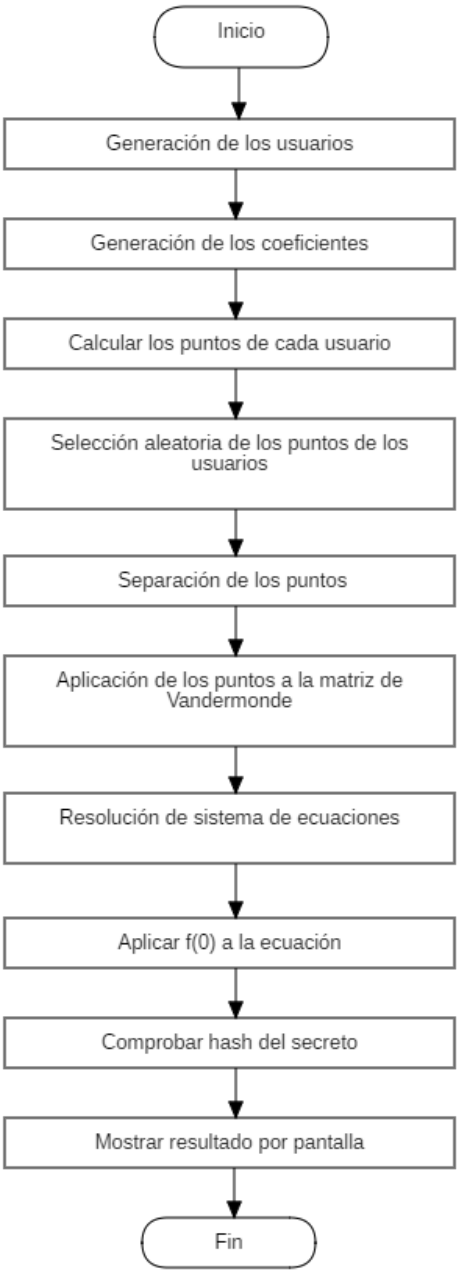

Figura 8: Diagrama de flujo Shamir

- **Diagrama de flujo de detecciones (Opción 3 y 4)**: Secuencias de la función detección de mentirosos con $\boldsymbol{k+1}$ (detección de 1 mentiroso) y $\boldsymbol{k+2}$ (detección de 2 mentirosos).

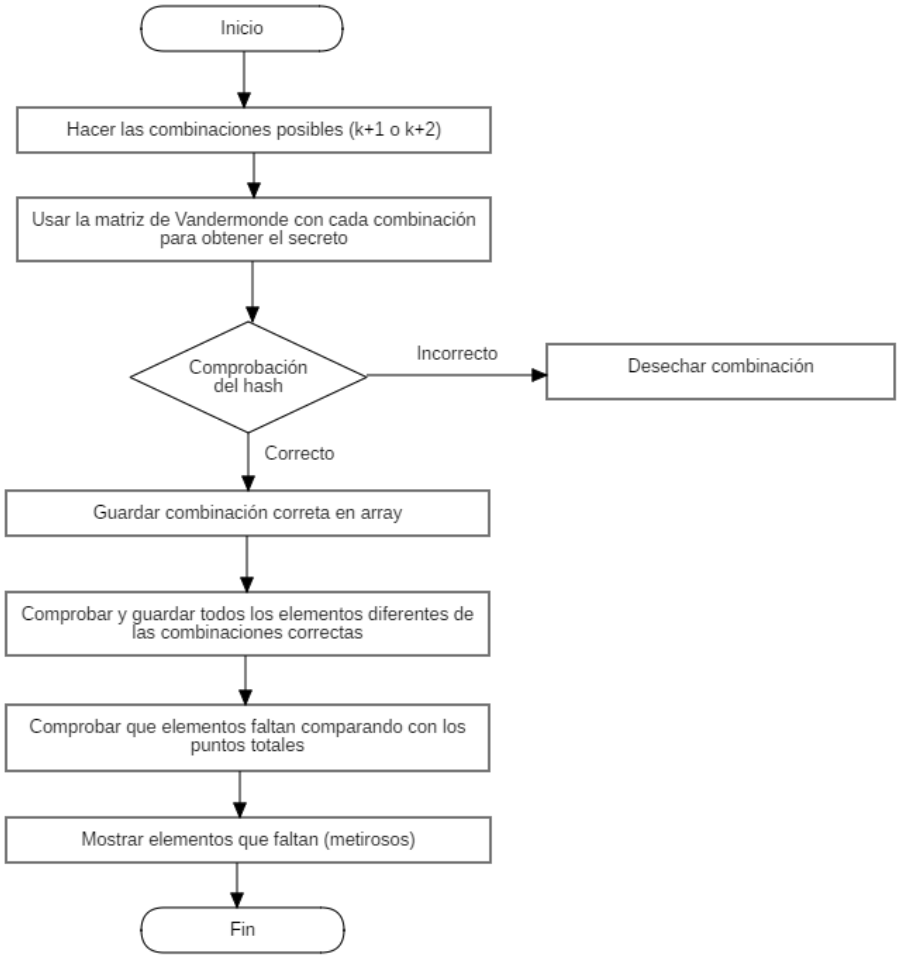

Figura 9: Diagrama de flujo de la función de detección de mentirosos

8. Estructura del código

En esta sección veremos las partes que tiene el código, los archivos que componen el proyecto y los métodos usados, dando detalles de su funcionamiento y de algunas técnicas usadas.

La carpeta del proyecto está formada por dos archivos, **'launcher.py'** y **'Shamir.py'**, ambos dentro del mismo directorio.

8.1. launcher.py

Actúa como menú de prueba para las funciones creadas en Shamir.py y su finalidad es invocar las funciones para poder simular el protocolo.

Funciona coma una 'interfaz' para pedir al usuario los datos necesarios y que este los introduzca para la invocación de las funciones.

Este archivo es el principal del proyecto, lo que quiere decir que es el que debemos ejecutar para iniciar el programa.

Al principio del archivo podemos observar que establecemos un campo de Galois (GF = galois.GF(2**64)). Esto es para cuando el programa recibe el secreto introducido por el usuario, convertirlo en cuerpo finito y poder trabajar con él sin problemas.

A continuación iremos comentando cada parte del código por separado.

8.1.1. Librerías, variables y menu()

A continuación vemos las librerías que se importan al archivo y las variables globales declaradas. Lo más destacable es la función menu(), que se encarga de mostrar al usuario las opciones disponibles y recoger la respuesta del usuario.

launcher.py - Librerías variables globales y menu()

```
import Shamir, galois, hashlib

GF = galois.GF(2**64)
n = None
S = None
t = None
hash= None
puntos = []

def menu():

    print("1.- Introducir datos")
    print("2.- Compartir mensaje")
    print("3.- Detección de 1 mentiroso")
    print("4.- Detección de 2 mentirosos")
    print("0.- Salir")
    print("————————————————————")
    opt = int(input("Introduce una opcion:"\n))

    return opt
```

8.1.2. Opciones 1 y 2

Estas opciones están dedicadas a la introducción de datos, como el número de usuarios, secreto a compartir o el umbral, y la simulación del protocolo de Shamir.

La opción 1 pregunta al usuario cuál es el número de usuarios que quiere crear, el secreto y el umbral. Por otra parte, el programa calculará el hash del secreto para usarlo más adelante.

En la opción 2 es la simulación del protocolo. Solo se podrá usar si previa-

mente se han introducido los datos en la opción 1. Esta se encarga de llamar a las funciones para calcular el secreto y mostrarlo por pantalla.

launcher.py - Opciones 1 y 2

```
if __name__ == "__main__":

    opcion = 1

    while(opcion != 0):
        opcion = int(menu())

        if(opcion == 1):
            n = int(input("¿Cuantos usuarios quieres crear?: "))
            S = GF(input("Numero secreto: "))
            hash = hashlib.sha256(str(S).encode())
            t = int(input('"Umbral: "))
            Shamir.comprobarUmbral(t,n)

        elif(opcion == 2):

            if n == None:

                print("Introduzca primeros los datos")
            else:

                puntos = Shamir.generacionpuntos(S,n,t))
                seleccion = Shamir.seleccionPuntos(puntos,t,n))
                xi, fi = Shamir.separacionPuntos(seleccion,t)
                Shamir.vandermonde(xi, fi, t, hash)
```

8.1.3. Opciones 3 y 4

Aquí probaremos las funciones específicas para la detección de los mentirosos. La opción 3 está ligada a la detección de 1 mentiroso y la 4 a la detección de 2 mentirosos.

Antes de la llamada a la función detección(), se hace una copia de los puntos, ya que se modifica uno de ellos para simular al mentiroso.

launcher.py - Opciones 3 y 4

```
elif(opcion == 3):

    puntos = Shamir.generacionpuntos(S,n,t)
    puntos[1]=[GF(2),GF(12312312312)]
    Shamir.deteccionK1(puntos, t, hash)

elif(opcion == 4):

    puntos = Shamir.generacionpuntos(S,n)
    puntos[1]=[GF(2),GF(12312312312)]
    puntos[2]=[GF(3),GF(12312312312)]
    Shamir.deteccionK2(puntos, t, hash)

    elif(opcion == 0):)
        print("Saliendo...")
    else:
        print("Opción no válida")
```

8.2. Shamir.py

En este archivo se encuentran todas las funciones que nos permiten simular la compartición del secreto y las mejoras propuestas mencionadas anteriormente. A continuación se irán viendo las funciones una a una para poder

explicarlas con más detalle.

8.2.1. Declaración de librerías y variables globales

Librerias y variables

```
import random
import numpy as np
from itertools import combinations
import galois
import hashlib

GF = galois.GF(2**64)
```

8.2.2. hashCheck

Una de las mejoras planteadas era la implementación de una función hash para asegurarnos de que el secreto fuese correcto y nadie haya hecho trampas, Esta función es la encargada de hacer ese trabajo.

Comprueba que el hash del secreto introducido por el usuario es el mismo que el del secreto calculado por los demás usuarios, devolviendo **True** o **False** y un mensaje por pantalla en función de si es correcto o no.

Esta función recibe como argumentos:

- **secreto**: Es el secreto calculado por los usuarios, que posteriormente en la función será calculado su hash correspondiente.
- **hash**: Hash del secreto introducido por el usuario.

hashCheck

```
def hashCheck(secreto, hash):

    secreto = hashing(secreto)

    if (secreto.hexdigest() == hash.hexdigest()):
        print("Secreto correcto")

        return True
    else:
        print("Secreto incorrecto")

        return False
```

8.2.3. format

Simplemente se encarga de tomar una cadena de texto y eliminar el primer y el último carácter de ella. La finalidad de esta función es, por una parte, por razones puramente estéticas. Adapta los resultados para mostrarlos de una manera más atractiva para el usuario, como se ve aquí:

- Sin formatear: El código secreto es:(12)
- Formateado: El código secreto es: 12

Esta modificación es apelativa para la otra motivación de uso, que sin la eliminación de, en este caso, los paréntesis conllevaría un error de interpretación del hash. Obtendríamos un resultado incorrecto, cuando no lo es, el del hash distinto a aquel correspondiente al secreto original.

format

```
def format(n):

    n = str(n)
    n = n[:-1]
    n = n[1:]

    return str(n)
```

8.2.4. hashing

Función que recibe un elemento y esta le aplica una función hash. En este caso se ha optado por SHA256.

hashing

```
def hashing(x):
    return(hashlib.sha256(str(x).encode()))
```

8.2.5. comprobarUmbral

Comprueba que el usuario no de un umbral(t) mayor que el número máximo de usuarios en el sistema(n). Si se diese el caso, el sistema le volvería a solicitar el umbral hasta que fuese un número menor.

comprobarUmbral

```
def comprobarUmbral(t,n):

    while(t >= n):
        print("Umbral incorrecto, debe de ser un numero menor al
numero de usuarios")
        t = int(input("Umbral: "))
```

8.2.6. generacionPuntos

Es la función encargada de calcular los puntos de cada usuario, desde la creación de todos los usuarios, generación de los coeficientes aleatorios hasta la valoración del polinomio en el punto del usuario.

Comienza con la generación de dos cuerpos finitos aleatorios y un array de usuarios vacío que se irá rellenando con un bucle, introduciendo cuerpos finitos de $\mathbf{1}$ hasta $\boldsymbol{n}$, Seguidamente se creará otro array para almacenar los coeficientes del polinomio, que serán los $\boldsymbol{k-1}$ números aleatorios generados y el secreto que queremos compartir. Una vez creado, usaremos la función "polyval" para evaluar el polinomio en los puntos indicados que, en este caso, son los usuarios, y se almacenarán los resultados en un array. Para finalizar, crearemos otro array para introducir los resultados obtenidos con cada usuario, quedando cada tupla del array de la siguiente manera:
[[usuario]**,** [valor del polinomio en el punto]].

generacionPuntos

```
def generacionpuntos(S,n,t):

    x = GF.Random(t-1)

    usuarios = []

    for i in range(n):
        usuarios.append(GF(i+1))

    coeficientes = []

    for i in range(t-1):
        coeficientes.append(x[i])

    coeficientes.append(s)

    resultados = []

    for i in range(n):
        resultados.append(np.polyval(coeficientes, usuarios[i]))

    puntos = []

    for i in range(len(usuarios)):
        puntos.append([usuarios[i],resultados[i]])

    print(puntos)

    return puntos
```

8.2.7. comprobacion

Es una función simple que comprueba que el usuario escogido no haya sido seleccionado con anterioridad. El motivo de esta función lo veremos en el siguiente apartado.

Comprueba que en el array exista el elemento que recibe, devolviendo True (si el elemento existe en el array) o False (si no existe en el array).

comprobacion

```
def comprobacion(X, usuarios):

    flag = True

    for i in range(len(usuarios)):
        if (usuarios[i] == x):
            flag = False
            break

    return flag
```

8.2.8. seleccionPuntos

Es la función encargada de seleccionar los puntos aleatoriamente para el cálculo del secreto.

Consiste en usar la función "randint" para crear un número aleatorio, que representará al usuario seleccionado. Antes de ser seleccionado se hará una comprobación de sí ese mismo usuario ha sido seleccionado en una iteración anterior, por lo que usaremos la función "comprobacion()" (explicada en el apartado anterior) para asegurarnos de que no se haya escogido ese usuario. Si ese usuario no ha sido escogido, se almacenará en el array usuarios. Una vez seleccionados todos los puntos, se almacenará en un array "seleccion"los

puntos correspondientes a cada usuario.

seleccionPuntos

```
def seleccionPuntos(puntos,t,n):

    usuarios = []
    seleccion = []

    i=0

    while i < t:
        x = random.randint(1,n)
        if comprobacion(x,usuarios):
            usuarios.append(x)
            i+=1

    for i in range(t):
        seleccion.append(puntos[usuarios[i] - 1])

    return seleccion
```

8.2.9. separacionPuntos

Una vez obtenidos los puntos seleccionados en una lista, esta función los separa en 2 listas diferentes, una para los usuarios (xi) y otra para sus valores en el polinomio (fi) para facilitar el trabajo más adelante.

Es en esta función donde se transformarán los valores de la lista fi en cuerpos finitos y la lista se convertirá en una matriz $\mathbf{3x1}$.

separacionPuntos

```
def separacionPuntos(puntos,t):

    xi=[]
    fi=[]

    for i in range (len(puntos)):
        xi.append((puntos[i][0]))

    for i in range (len(puntos)):
        fi.append((puntos[i][1]))

    fi2 = GF([fi])
    fi2.shape = (t,1):

    return xi, fi2
```

8.2.10. vandermonde

Es en esta función donde se obtendrá el secreto. Como ya se mencionó antes, usaremos la matriz de Vandermonde para calcularlo. Primero crearemos una matriz llena de ceros de las dimensiones (t,t) y la rellenaremos con los elementos del array "xi" (recordemos que estos elementos son los usuarios). Posteriormente, usaremos la función de NumPy "linalg.solve()" con la matriz previamente transformada fi para obtener el polinomio y la aplicaremos $\boldsymbol{f(0)}$ para recuperar el secreto.

Una vez obtenido el secreto, emplearemos la función hashchek para comprobar si el hash coincide con el del secreto original.

vandermonde

```
def vandermonde(xi,fi,t,hash):

    M = GF.Zeros((t,t))

    for i in range(t):
        for j in range(t):
            k = (t-1)-j
            M[i,j] = GF(xi[i]**k)

    resultado = np.linalg.solve(M,fi)

    print("El codigo secreto es: " + (format(resultado[t-1])))

    return hashCheck(format(resultado[t-1]), hash)
```

8.2.11. deteccion

Es una de las mejoras propuestas. Se encarga de detectar a los tramposos aumentando el umbral e ir probando las diferentes combinaciones de usuarios para poder ver en cuál de todas ellas el secreto no es correcto y buscar el usuario común en todas esas combinaciones.

Usaremos la función "combinations" de la librería itertools para crear todas las combinaciones posibles de los puntos y aplicaremos las funciones de separacionPuntos y vandermonde para calcular el secreto.

Si el secreto calculado con esa combinación es correcto, es decir, el hash del secreto coincide con el hash del secreto original, la combinación se guardará en un array "combinacionesCorrectas" para que, una vez que se hayan calculado todos los secretos con todas las combinaciones y ver cuáles nos dan el secreto correcto, comprobaremos que elementos diferentes están en esas combinaciones para almacenarlas en otro array "exclusión". Esto servirá para compararlo con el array "puntos" y ver que los elementos que estén en "puntos", y no en "exclusion", serán los mentirosos.

deteccion (con umbral $\boldsymbol{k+1}$ para la detección de un mentiroso)

```
def deteccionK1(puntos, t, hash):

    combinacionesCorrectas=[]
    exclusion=[]
    mentirosos=[]

    for combinacion in combinations(puntos, t+1):

        xi, fi = separacionPuntos(combinacion,t+1)

        if(vandermonde(xi, fi, t+1, hash)):
            combinacionesCorrectas.append(combinacion)

    for i in range (len(combinacionesCorrectas)):
        for elemento in combinacionesCorrectas[i]:
            if elemento not in exclusion:
                exclusion.append(elemento)

    for elemento in puntos:
        if elemento not in exclusion:
            mentirosos.append(elemento)

    print("El mentiroso es: ")
    for mentiroso in mentirosos:
        print(mentiroso)
```

deteccion (con umbral $\boldsymbol{k+2}$ para la detección de dos mentirosos)

```
def deteccionK2(puntos, t, hash):

    combinacionesCorrectas=[]
    exclusion=[]
    mentirosos=[]

    for combinacion in combinations(puntos, t+2):

        xi, fi = separacionPuntos(combinacion,t+2)

        if(vandermonde(xi, fi, t+2, hash)):
            combinacionesCorrectas.append(combinacion)

    for i in range (len(combinacionesCorrectas)):
        for elemento in combinacionesCorrectas[i]:
            if elemento not in exclusion:
                exclusion.append(elemento)

    for elemento in puntos:
        if elemento not in exclusion:
            mentirosos.append(elemento)

    print("Los mentirosos son: ")
    for mentiroso in mentirosos:
        print(mentiroso)
```

9. Manual de usuario

En esta sección se explicará como usar el software creado para poder realizar las pruebas posteriormente.

Para ejecutar la aplicación, deberemos importar el código a un editor de código y presionar el botón de ejecución.

Una vez iniciada se desplegará un menú con las siguientes opciones:

Menú

```
1.- Introducir datos
2.- Compartir mensaje
3.- Detección de 1 mentiroso
4.- Detección de 2 mentirosos
0.- Salir
----------------------------------------
Introduce una opcion:
```

Para seleccionar una opción se debe introducir su respectivo número, pero para que funcione correctamente, como primera opción siempre debe ser **1. Introducir datos**. Aquí se recogerán el secreto, el umbral y el número de usuarios con los que queremos hacer las pruebas.

Si seleccionamos una opción diferente antes que la 1, el sistema avisará de que primero se deben introducir los datos.

Una vez seleccionada la primera opción, deberemos proporcionar los datos que queremos:

Opción 1

```
Introduce una opcion:
1
¿Cuantos usuarios quieres crear?: 20
Numero secreto: 5678
Umbral: 7
```

Y una vez introducidos, podremos usar cualquiera de las 3 opciones siguientes.

9.1. Compartir mensaje

Esta opción simulará el protocolo de Shamir con los datos que el usuario introdujo en la primera opción. Al iniciarla el sistema hará los cálculos necesarios y al final mostrará el resultado final con un mensaje de si es correcto o no.

Opción 2

```
El codigo secreto es: 5678
Secreto correcto
```

9.2. Detección de mentirosos

Las siguientes opciones son prácticamente la misma función, solo cambiando el detalle de la suma que se hace al umbral (**+1 opción 3 y +2 opción 4**) para la detección del o los usuarios mentirosos. Al ejecutar cualquiera de las 2 opciones, el sistema detectará al mentiroso y mostrándolo por pantalla.

Es importante ejecutar la opción 2 previamente para que los puntos sean creados, ya que sin ellos, la opción dará un error.

Opción 3

```
El mentiroso es:
[GF(2, order=2^64), GF(12312312312, order=2^64)]
```

Opción 4

```
Los mentirosos son:
[GF(2, order=2^64), GF(12312312312, order=2^64)]
[GF(3, order=2^64), GF(12312312312, order=2^64)]
```

9.3. Salir

La opción 0 es simplemente una opción para terminar las pruebas y salir del programa.

10. Pruebas

En esta sección realizaremos las pruebas para comprobar el correcto funcionamiento del código, acompañadas de capturas de pantalla en el que explicaremos el resultado y el motivo de los datos introducidos.

10.0.1. Pruebas del protocolo Shamir

Para empezar, nos centraremos en el funcionamiento del protocolo. Como primera prueba introduciremos los datos usando la opción 1 del menú y veremos si el resultado es correcto.

En este caso habrá **10 usuarios**, el secreto a compartir será **12** y un umbral de **5**.

Introducción de datos

```
Introduce una opcion:
1
¿Cuantos usuarios quieres crear?: 10
Numero secreto: 12
Umbral: 5
```

Una vez introducidos los datos, iniciaremos la simulación eligiendo la opción 2. Obtendremos los siguientes resultados:

Resultados 1

```
El codigo secreto es: 12
Secreto correcto
```

Se trata del secreto calculado y la comprobación del hash. Podemos comprobar que se ha obtenido correctamente el secreto. Ahora comprobaremos

que se obtienen los resultados correctos con muchos más usuarios.

Ahora tendremos **100 usuarios**, el número secreto será **71271** y habrá un **umbral de 20**.

Introducción de datos 2

```
Introduce una opcion:
1
¿Cuantos usuarios quieres crear?: 100
Numero secreto: 71271
Umbral: 20
```

Y los resultados son:

Resultados 2

```
El codigo secreto es: 71271
Secreto correcto
```

Volvemos a tener los resultados correctos. Podemos ver que la simulación del protocolo funciona sin ningún problema.

Ahora probaremos si es capaz de detectar que el secreto no es el correcto y comprobar que compara los hash correctamente. Para ello, modificaremos los valores de un usuario aleatorio para que salga un secreto incorrecto y usaremos los mismos datos que antes:

Dato incorrecto

```
puntos = Shamir.generacionpuntos(S,n)

seleccion = Shamir.seleccionPuntos(puntos,t,n)

seleccion[1][1] = GF(1000)

xi, fi = Shamir.separacionPuntos(seleccion,t)

Shamir.vandermonde(xi, fi, t, hash)
```

Como los usuarios se eligen aleatoriamente, tendremos que editar el punto directamente desde el código, cogiendo una posición que queramos y cambiar su punto. Esto es porque si cambiamos el punto y el usuario, el programa podrá saltar un error si hay dos usuarios iguales.

Introducción de datos 3

```
Introduce una opcion:
1
¿Cuantos usuarios quieres crear?: 100
Numero secreto: 71271
Umbral: 20
```

Resultados 3

```
El codigo secreto es: 174902824549332277 17
Secreto incorrecto
```

Vemos que el programa detecta que el secreto es incorrecto y avisa al usuario. En el siguiente apartado probaremos la forma para detectar al mentiroso que ha aportado los datos incorrectos.

A continuación haremos más pruebas para comprobar el comportamiento y los resultados del algoritmo.

Introducción de datos 4

```
¿Cuantos usuarios quieres crear?: 70
Numero secreto: 678873
Umbral: 50
```

Resultados 4

```
El codigo secreto es: 678873
Secreto correcto
```

Introducción de datos 5

```
¿Cuantos usuarios quieres crear?: 30
Numero secreto: 78978
Umbral: 3
```

Resultados 5

```
El codigo secreto es: 78978
Secreto correcto
```

Introducción de datos 6

```
¿Cuantos usuarios quieres crear?: 20
Numero secreto: 12345
Umbral: 15
```

Resultados 6

```
El codigo secreto es: 12345
Secreto correcto
```

Introducción de datos 7

```
¿Cuantos usuarios quieres crear?: 50
Numero secreto: 45674616
Umbral: 45
Punto alterado: seleccion[1][1] = GF(9043089)
```

Resultados 7

```
El codigo secreto es: 12352046020154903451
Secreto incorrecto
```

Introducción de datos 8

```
¿Cuantos usuarios quieres crear?: 40
Numero secreto: 456456456
Umbral: 10
Punto alterado: seleccion[1][1] = GF(51)
```

Resultados 8

```
El codigo secreto es: 184235840831866113 27
Secreto incorrecto
```

Introducción de datos 9

```
¿Cuantos usuarios quieres crear?: 80
Numero secreto: 456123
Umbral: 15
Punto alterado: seleccion[1][1] = GF(1234)
```

Resultados 9

```
El codigo secreto es: 7941804170662537370
Secreto incorrecto
```

Introducción de datos 10

```
¿Cuantos usuarios quieres crear?: 10
Numero secreto: 45389
Umbral: 8
Punto alterado: seleccion[1][1] = GF(585646)
```

Resultados 10

```
El codigo secreto es: 13214503264759491183
Secreto incorrecto
```

10.0.2. Pruebas de detección de mentirosos

Ahora pasaremos a probar el funcionamiento de las funciones de detección. Para ello veremos por separado las dos funciones (detección con $\boldsymbol{k+1}$ y detección con $\boldsymbol{k+2}$).

En cada una de las pruebas se establecerán la cantidad de mentirosos a detectar según la función que estemos probando, es decir, si es detección $\boldsymbol{k+1}$ y si es detección $\boldsymbol{k+2}$ habrá 2 mentirosos.

Para empezar, usaremos los siguientes datos:

Introducción de datos 11

```
Introduce una opcion:
1
¿Cuantos usuarios quieres crear?: 10
Numero secreto: 791834
Umbral: 4
```

Ahora comprobaremos si el programa es capaz de detectar al mentiroso. En este escenario se ha creado uno que será el **usuario 2 con el punto 12312312312**, que suponemos que ha modificado su punto para dar uno incorrecto.

Detección de un mentiroso

```
El mentiroso es:
[GF(2, order=2^64), GF(12312312312, order=2^64)]
```

Podemos ver que es detectado sin ningún problema. Como ya se mencionó anteriormente, la prueba puede extenderse bastante en el tiempo en función del **n.º de usuarios y el umbral** dado. En este caso, da el resultado en poco tiempo.

Con los mismos datos introducidos, ahora intentaremos la detección de 2 usuarios mentirosos, que serán los **usuarios 2 y 3 con el punto 12312312312**:

Detección de dos mentirosos

```
Los mentirosos son:
[GF(2, order=2^64), GF(12312312312, order=2^64)]
[GF(3, order=2^64), GF(12312312312, order=2^64)]
```

Como conclusión, vemos que se han obtenido correctamente todos los resultados, destacando que la detección de mentirosos, sin haber hecho la prueba del escenario de los tramposos poniéndose de acuerdo entre sí para dar un punto compatible al ser un caso imposible debido a que los puntos se

calculan antes de ser seleccionados aleatoriamente y suponemos que existe un servidor externo que revele todos los puntos de forma simultánea.

Como hemos hecho anteriormente, a continuación habrá más pruebas realizadas.

Introducción de datos 11

```
¿Cuantos usuarios quieres crear?: 10
Numero secreto: 6564
Umbral: 6
Tramposo: puntos[6]=[GF(7),GF(18)]
```

Resultados 11

```
El mentiroso es:
[GF(7, order=2^64), GF(18, order=2^64)]
```

Introducción de datos 12

```
¿Cuantos usuarios quieres crear?: 15
Numero secreto: 465456
Umbral: 10
Tramposo: puntos[3]=[GF(4),GF(45456465456)]
```

Resultados 12

```
El mentiroso es:
[GF(4, order=2^64), GF(45456465456, order=2^64)]
```

Introducción de datos 13

```
¿Cuantos usuarios quieres crear?: 15
Numero secreto: 465456
Umbral: 3
Tramposo: puntos[7]=[GF(8),GF(73642)]
```

Resultados 13

```
El mentiroso es:
[GF(8, order=2^64), GF(73642, order=2^64)]
```

Introducción de datos 14

```
¿Cuantos usuarios quieres crear?: 10
Numero secreto: 5646
Umbral: 6
Tramposos: puntos[1]=[GF(2),GF(56346)]
           puntos[2]=[GF(3),GF(793133)]
```

Resultados 14

```
Los mentirosos son:
[GF(2, order=2^64), GF(56346, order=2^64)]
[GF(3, order=2^64), GF(793133, order=2^64)]
```

Introducción de datos 15

```
¿Cuantos usuarios quieres crear?: 20
Numero secreto: 456564456645
Umbral: 16
Tramposos: puntos[2]=[GF(3),GF(9342861)]
           puntos[4]=[GF(5),GF(18634)]
```

Resultados 15

```
Los mentirosos son:
[GF(3, order=2^64), GF(9342861, order=2^64)]
[GF(5, order=2^64), GF(18634, order=2^64)]
```

Introducción de datos 16

```
¿Cuantos usuarios quieres crear?: 15
Numero secreto: 56
Umbral: 10
Tramposos: puntos[10]=[GF(11),GF(73194682)]
           puntos[4]=[GF(5),GF(28643)]
```

Resultados 16

```
Los mentirosos son:
[GF(5, order=2^64), GF(28643, order=2^64)]
[GF(11, order=2^64), GF(73194682, order=2^64)]
```

11. Conclusiones y propuestas de mejora

Una vez terminado el proyecto, analizaremos si se han cumplido los objetivos planteados al principio y se hará un breve resumen de ellos, considerando todo lo aprendido en el proyecto.

11.1. En el ámbito del proyecto

- Se han implementado métodos que hacen al protocolo más seguro.
- Se ha conseguido detectar a los tramposos que envían puntos incorrectos.
- Se han planteado otros métodos para reforzar la seguridad que se pueden implementar en un futuro.

11.2. En el ámbito personal

- He conocido y entendido el funcionamiento del protocolo de Shamir. A su vez se han conseguido muchos conocimientos sobre criptografía y su aplicación en la seguridad informática.
- Se ha mejorado grado de aprendizaje sobre Python.
- He descubierto nuevas librerías y herramientas que pueden ser útiles para un futuro.

Dicho esto, podemos decir que todos los objetivos del proyecto se han completado satisfactoriamente, además de llevarme conmigo muchos conocimientos nuevos que he disfrutado conocer e investigar sobre ellos.

En este proyecto se han presentado diferentes métodos para la detección de tramposos, pero aún se puede seguir investigando para encontrar más métodos.

- **Implementación de un servidor con "confiable"**. Será el encargado de hacer los cálculos de los puntos paralelamente con los usuarios, de forma que si detecta que si su hash es distinto al del secreto calculado por los usuarios, este avisará a los demás usuarios de quien es el tramposo y no revelará el secreto hasta que no haya errores.
- **Implementación de firmas digitales** en los puntos que envían los usuarios para asegurarse de que los puntos no puedan ser modificados en el envío de datos al servidor, o que un tramposo intente pasar por alguien de confianza y pueda obtener el secreto.
- **Estudiar formas de mejora el algoritmo de detección de usuarios** para que tarde menos en detectar a los tramposos.

Todas estas propuestas son planteadas para nuestro proyecto, pero también se podría ir más lejos estudiando otro tipo de protocolos con la finalidad de crear no solo una solución para Shamir, sino para una lista amplia de protocolos con alta probabilidad de ser vulnerados.

Referencias

[1] Alfredo Sánchez Alberca. *La librería Numpy*. 2022. URL: https://aprendeconalf.es/docencia/python/manual/numpy/.

[2] Maria Arce Muela et al. "Reparto de secretos". En: (2016), págs. 3-6.

[3] Andoni Arco. *Lenguaje de alto nivel en programación: Definición y tipos*. 2022. URL: https://miformacion.eu/blog/lenguaje-de-alto-nivel-programacion/.

[4] Varios autores. *¿Qué es la criptografía?* 2023. URL: https://aws.amazon.com/es/what-is/cryptography/.

[5] Varios autores. *¿Qué es Python?* 2023. URL: https://aws.amazon.com/es/what-is/python/.

[6] Varios autores. *Criptografía de clave asimétrica*. URL: https://www.cert.fnmt.es/curso-de-criptografia/criptografia-de-clave-asimetrica.

[7] Varios autores. *Esquema de Shamir*. URL: https://www.wikiwand.com/es/Esquema_de_Shamir.

[8] Varios autores. *itertools — Functions creating iterators for efficient looping*. 2023. URL: https://docs.python.org/3/library/itertools.html.

[9] Varios autores. *itertools — Functions creating iterators for efficient looping*. 2023. URL: https://docs.python.org/3/library/itertools.html.

[10] Varios autores. *Protocolo de tres envíos*. 2020. URL: https://es.wikipedia.org/wiki/Protocolo_tres_env%C3%ADos.

[11] Varios autores. *Python*. 2023. URL: https://es.wikipedia.org/wiki/Python.

[12] Varios autores. *random —Generar números pseudoaleatorios*. 2023. URL: https://docs.python.org/es/3/library/random.html.

[13] GuilleVen. *¿Qué es la criptografía?* URL: https://www.tecnologia-informatica.com/que-es-la-criptografia/#Algoritmos_criptogr%C3%A1ficos.

[14] Matt Hostetter. *Galois: A performant NumPy extension for Galois fields*. Nov. de 2020. URL: https://github.com/mhostetter/galois.

[15] Javier Sáez Hurtado. *Qué es la criptografía y para qué sirve*. 2022. URL: https://www.iebschool.com/blog/que-es-la-criptografia-y-para-que-sirve-finanzas/.

[16] Heather Woll Martin Tompa. "How to share a secret with cheaters". En: (1998), págs. 261-264.

[17] Eugenia Monforte. *Función hash: ¿qué es y cómo se usa en la firma electrónica?* URL: https://www.camerfirma.com/funcion-hash-que-es-y-como-se-usa-en-la-firma-electronica/.

[18] Javier Pastor. *¿Qué es un hash?* 2023. URL: https://academy.bit2me.com/que-es-hash/.

[19] A Vázquez-Ingelmo y FJ Garcıa-Peñalvo. "Introducción al Proceso Unificado". En: (2019), págs. 11-16.

[20] Jorge Luis Villar Santos, Carles Padró Laimón y Germán Sáez Moreno. "Compartición de secretos en criptografıa". En: *Buran* 10 (1997), págs. 53-55.