UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ESQUEMA DE COMPARTICIÓN DE SECRETOS CON DETECCIÓN EFICIENTE DE TRAMPAS

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:

DANIEL BECERRA PEDRAZA

DIRECTOR DE LA TESIS:

DR. GERARDO VEGA HERNÁNDEZ
POSGRADOENCIENCIAEINGENIERÍADELACOMPUTACIÓN

CIUDAD UNIVERSITARIA, CD. MX., SEPTIEMBRE DE 2019

Resumen

En el presente trabajo se realiza un estudio de los esquemas de compartición de secretos basados en la interpolación de Lagrange, partiendo del esquema básico hasta aquellos con detección eficiente de trampas.

Se presenta un breve análisis realizado a algunos esquemas recientes con detección eficiente de trampas resaltando algunos puntos donde se presenta un comportamiento que difiere con el reportado para dichos esquemas. Adicionalmente se resumen sus propiedades y se mencionan algunas mejoras que pueden realizarse en sus algoritmos.

Se define un esquema basado en los esquemas analizados, mismo que conserva sus ventajas y el cual sí alcanza el comportamiento buscado en dichos esquemas. Además, la función verificadora del esquema propuesto utiliza una técnica de descomposición que permite una detección de trampas más eficiente en la mayoría de los casos.

Posteriormente se realiza una comparación de las propiedades de los esquemas analizados y las del esquema propuesto.

Finalmente, para ilustrar el comportamiento de ambos esquemas, se incluyen los resultados de algunas pruebas realizadas al desempeño de la implementación de uno de los esquemas analizados versus el desempeño del esquema propuesto en el presente trabajo.

Agradecimientos

Principalmente quiero agradecer a mi asesor, el Dr. Gerardo Vega Hernández por su esfuerzo, su apoyo, sus enseñanzas, el tiempo valioso que invirtió en mí, por motivarme a aprovechar oportunidades y en especial, por su paciencia.

A mis sinodales, por su apoyo, sus amables comentarios, observaciones y sugerencias, así como por el tiempo valioso dedicado a la revisión de este trabajo.

A mis padres y hermanos por el apoyo que me han brindado este tiempo a pesar de la distancia.

A Arturo, por confiar en mí, por su apoyo, por seguir adelante a pesar de los problemas en el camino y sobre todo por mantener vivo nuestro proyecto estos dos últimos años.

A Iris y Jacob, por su compañía, su apoyo y por aguantarme a mi y a Chimuelo todo un año.

A Jaime y su familia, por haberme apoyado en todo momento y animarme a seguir adelante.

A Víctor, por su amistad incondicional, por creer y confiar en mi, por su apoyo y por seguir adelante ante todo.

A mis profesores Sergio, Antonio, Adrián, Alin, Marco y Jorge Arturo, por haberme apoyado y motivado a seguir adelante en momentos cruciales y haber extendido sus enseñanzas más allá de las cuatro paredes del aula.

A mis compañeros, profesores y comunidad del posgrado por su compañía, sus enseñanzas y más que nada por su increíble apoyo.

Al resto de mis amigos, quienes han creído en mi y me han apoyado a pesar de la distancia.

Al PAPIIT-UNAM IN109818 por el apoyo recibido para la realización de este trabajo así como para el artículo derivado del mismo.

y finalmente, al CONACyT por el apoyo brindado estos últimos dos años.

Índice general

Índice de figuras

Índice de cuadros

Prefacio

Supóngase, como en el problema que plantea G. J. Simmons en [3], que para realizar una transacción en cierta institución bancaria, si el monto es lo suficientemente grande, ésta deberá ser autorizada ya sea por tres gerentes, o bien dos vicepresidentes, o incluso por dos gerentes y un vicepresidente. Considérese ahora el caso planteado por C. Liu en [4] en que un grupo de investigadores que trabaja en un proyecto secreto desea resguardar sus documentos en una caja fuerte que sólo puede ser abierta si y sólo si al menos cierto número de ellos está presente.

Físicamente, estos problemas pueden arreglarse de diversas formas, por ejemplo, para el grupo de investigadores, si se requieren al menos tres de ellos para acceder a la caja fuerte, entonces se instala en la misma una cerradura que sólo pueda ser abierta si se ingresan al menos tres llaves válidas.

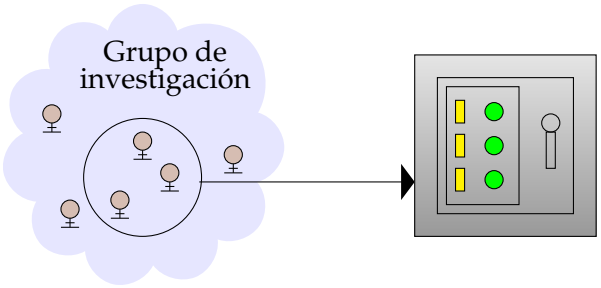

Figura 0-1: Intento de acceso a la caja fuerte con resultado satisfactorio.

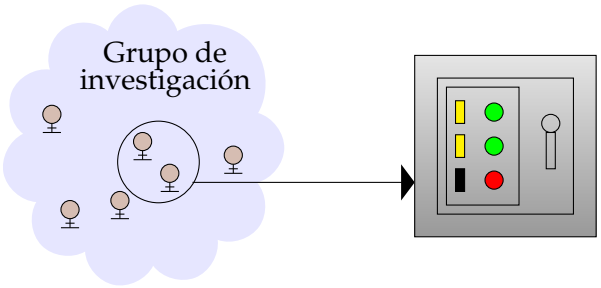

Figura 0-2: Intento de acceso a la caja fuerte con resultado insatisfactorio.

Los escenarios mencionados anteriormente pueden tener variantes, como el que haya más de una caja fuerte y cada científico deba tener una sola llave, que el banco contrate o despida empleados de los cuales dependa la autorización de las transacciones, etc. Mientras más condiciones tenga el problema, mayor será la complejidad de la solución requerida.

En la actualidad, la información es principalmente manejada a través de computadoras, básicamente como secuencias de bits. Dichas secuencias pueden ser interpretadas como números, por lo que una solución para este tipo de problemas abordada desde el punto de vista matemático y computacional es ideal.

Los esquemas de compartición de secretos, propuestos independientemente en 1979 por A. Shamir [5] y G.R. Blakley [6], son protocolos criptográficos que ofrecen una solución adecuada a problemas como los descritos anteriormente, es decir, en los cuales se desea proteger cierta información (un secreto) a través de un conjunto de entidades (participantes). Lo anterior se logra compartiendo a dichas entidades información (partes o sombras) aparentemente independiente del secreto, de tal forma que sólo subconjuntos autorizados (conjuntos de acceso) de las entidades puedan acceder al contenido del secreto y que los demás subconjuntos no puedan saber nada de él. Estas soluciones permiten incluir altos niveles de confidencialidad y de confianza.

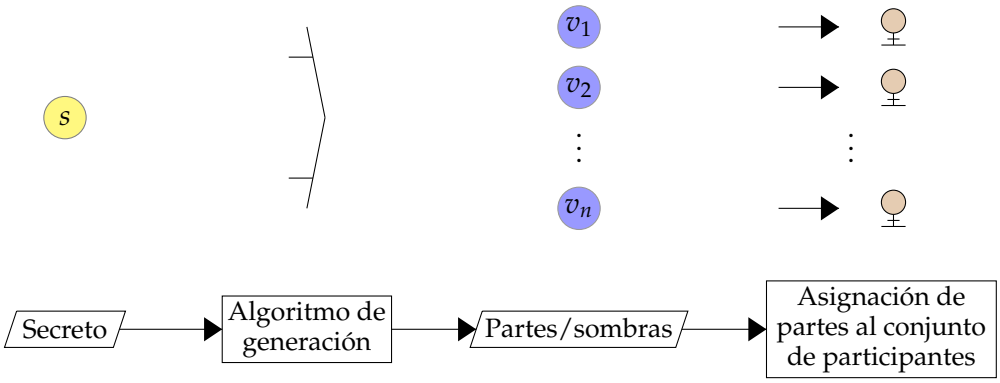

Figura 0-3: Proceso básico de compartición de un secreto.

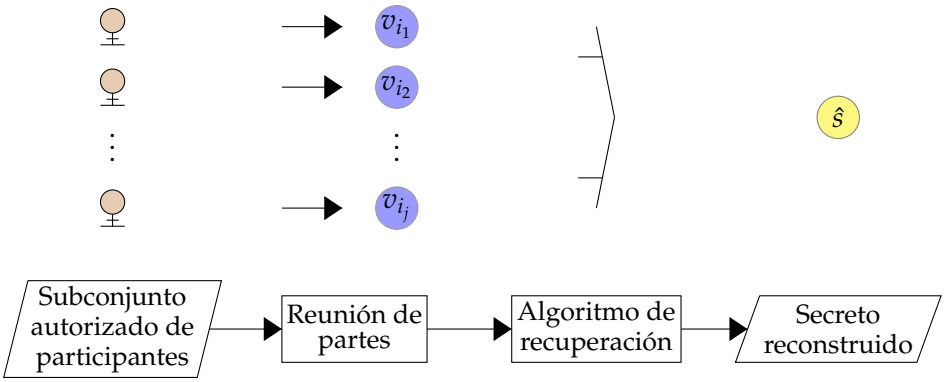

Figura 0-4: Proceso básico de recuperación de un secreto.

Un esquema de compartición de secretos básico permite, por ejemplo, resguardar un dato en múltiples servidores de tal forma que si un atacante compromete alguno de ellos no sabría nada del dato resguardado, incluso si algunos servidores fallan, el dato resguardado aún podría ser recuperado.

En las últimas décadas se han estudiado diversos escenarios y problemas que pudiesen presentarse durante la implementación de estos protocolos con el fin de enriquecer su comportamiento. Uno de estos problemas es el descrito por M. Tompa y H. Woll en [7]. Ellos analizaron el escenario donde uno o más participantes (tramposos) de un conjunto de acceso modifican sus partes para evitar, con una probabilidad de éxito ϵ, que los participantes honestos puedan recuperar el secreto.

Para ilustrar lo anterior, supóngase ahora, similar al ejemplo en [8, pág. 71], que la contraseña para autorizar el lanzamiento de un misil nuclear fue resguardada entre varios generales mediante un esquema de compartición de secretos sencillo, además, se sabe que entre los generales existe al menos un traidor, quien, si es parte del conjunto autorizado para recuperar el secreto, buscará por cualquier medio recuperar la contraseña para sí mismo e impedir al mismo tiempo que los demás puedan acceder a ella.

Una forma en que dicho traidor puede hacer esto es alterando la *parte/sombra* que se le asignó para que en el proceso de recuperación del secreto, la contraseña recuperada sea falsa. Si el esquema ocupado para compartir el secreto es un esquema básico, el traidor puede impedir que los demás miembros del conjunto de acceso recuperen la contraseña. Incluso, como se ejemplifica en [9, págs. 93–95], dependiendo de las herramientas matemáticas en que se base el esquema usado, de sus propiedades, así como de los intentos que se lleven a cabo para recuperar dicha contraseña, es posible que el mencionado traidor pueda conocer la contraseña real de una manera sencilla. Para evitar que dicho traidor pueda recuperar la contraseña y tenga así la libertad de lanzar el misil a su antojo, se requiere mejorar el esquema básico para que durante el proceso de recuperación se detecte si la información proporcionada por los miembros del conjunto de acceso fue alterada y si es el caso no se revele la información resguardada.

El esquema que presentaron M. Tompa y H. Woll en [7], el cual permite detectar si algún participante del conjunto de acceso modificó su parte, está basado en el esquema de A. Shamir [5]. Sin embargo, ideas como las expuestas en [10, 11] y esquemas como los presentados en [12, 13], han tenido un impacto importante en la compartición de secretos, pues utilizan herramientas distintas a las del esquema en [5] para compartir y recuperar un secreto.

Estas ideas y esquemas han inspirado una gran variedad de propuestas de esquemas de compartición de secretos, los cuales son relevantes por sus propiedades, incluso algunos de ellos abordan el escenario identificado por M. Tompa y H. Woll como es el caso de [14, 15, 16, 17, 18]. Sin embargo, como se espera que estos protocolos sean implementados propiamente en computadoras, la atención se centra en aquellos esquemas que son fácilmente adaptables a los problemas estudiados, así como también en aquellos cuya implementación sea lo más simple y eficiente posible.

Los esquemas de compartición de secretos básicos, así como aquellos con detección de trampas se han aplicado a varios problemas: Sistemas de votación electrónica [19, 20], protección de datos sensibles [21, 22], definición de criptosistemas [23], almacenamiento en la nube [24, 25, 26], autenticación en IoT (Internet of Things) [27, 28, 29], entre otros.

Algunos esquemas de compartición de secretos con detección de trampas también pueden

identificar a los tramposos [14, 17], o permitir que los participantes verifiquen que las partes de los demás participantes sean correctas. [30, 31], incluso compartir más de un secreto a la vez [15], etc.

Los esquemas de compartición de secretos con detección de trampas basados en el esquema de A. Shamir han sido ampliamente estudiados, por lo que se han modelado a los participantes tramposos de acuerdo con lo que podrían saber sobre el secreto [32, 33], incluso la cantidad de información que se requiere compartir ha sido acotada para ambos modelos en [33]. Los esquemas como los presentados en [1], [34] y [35] son relevantes debido a su rendimiento eficiente en la detección de trampas, esta eficiencia reside en la forma en que *descomponen* el secreto para generar un dígito verificador, a través del cual se llevará a cabo la detección de trampas. Además, la estructura principal de estos esquemas permite proponer nuevos esquemas eficientes y que sean capaces de compartir más secretos de manera óptima.

El objetivo principal del presente trabajo es el de definir un esquema que aproveche las ventajas de los esquemas en [1], [34] y [35] y que funcione de acuerdo a lo esperado para este tipo de esquemas.

El presente trabajo se estructura de la siguiente manera: En el Capítulo 1 se describen los esquemas de compartición de secretos relevantes para la propuesta desarrollada, partiendo del esquema básico, incluyendo aquellos con detección de trampas y finalizando con la descripción de unos esquemas recientes, los cuales son notables por tener una detección eficiente de trampas. En el Capítulo 2 se describe el esquema propuesto, se realiza una comparación de dicho esquema con los esquemas presentados en el Capítulo 1, la primer comparación es en cuanto a sus propiedades y la siguiente comparación es en cuanto al desempeño de la implementación de estos esquemas sometidos a algunas pruebas. Por último se presentan las conclusiones y se incluyen 2 apéndices, el Apéndice A corresponde al código fuente de los programas realizados y el Apéndice B a las evidencias de las actividades derivadas del presente trabajo, las cuales se describen a continuación:

- La impartición de la conferencia *¿Cómo compartir un secreto con tramposos?* el miércoles 7 de noviembre del 2018 en el simposio "Seguridad en Cómputo: Una Visión Académica" celebrado en el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas en Ciudad Universitaria.
- La realización del artículo *Secret Sharing Scheme With Efficient Cheating Detection*, presentado el jueves 28 de marzo del presente año en "The 2nd International Conference on Networking, Information Systems & Security" celebrado en Rabat, Marruecos. Este artículo ganó el premio al segundo mejor artículo del evento, fue publicado por la Association for Computing Machinery y está disponible en su biblioteca digital.
- La impartición de la conferencia *Esquema de Compartición de Secretos con Detección Eficiente de Trampas* el viernes 26 de abril del presente año en el "13 Coloquio Nacional de Códigos, Criptografía y Áreas Relacionadas" celebrado en el Palacio de Minería en la Ciudad de México.

Objetivos

Definir un esquema de compartición de secretos con detección eficiente de trampas en el cual la información compartida sea la mínima necesaria, que aproveche las ventajas de los esquemas en [34], [35] y [1] y que funcione de acuerdo a lo reportado para dichos esquemas.

Realizar una implementación del esquema propuesto y comparar su funcionamiento con el del esquema en [1].

Ejemplificar el uso del esquema propuesto mediante una posible aplicación.

Capítulo 1

Compartición de secretos

En este capítulo se presentan los *esquemas de compartición de secretos*, especialmente aquellos que están basados en la interpolación de Lagrange. Se describen sus características básicas hasta aquellas que permiten una detección eficiente de trampas, finalmente se incluyen dos *esquemas de compartición de secretos con detección eficiente de trampas* propuestos recientemente para los cuales se realiza un breve análisis.

1.1. Aspectos preliminares

El objetivo principal de un *esquema de compartición de secretos* es el de resguardar cierta información (un *secreto*) a través de un conjunto de entidades (*participantes*). Esto se logra compartiendo con dichas entidades cierta información (*partes* o *sombras*) aparentemente independiente del secreto, de tal forma que sólo subconjuntos autorizados (conjuntos de acceso) de las entidades puedan acceder al contenido del secreto y que los demás subconjuntos no puedan saber nada de él. Una definición general es la que se presenta a continuación:

Definición 1.1.1. [36] Un *esquema de compartición de secretos* consiste de:

- Un repartidor D;
- Un grupo de n participantes, $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$;
- Un espacio de secretos $\mathcal{S}$;
- n espacios de intercambio $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n$;
- Un procedimiento de compartición de secretos; y
- Un proceso de recuperación de secretos.

El repartidor D escoge un secreto $s \in \mathcal{S}$ y emplea un algoritmo de compartición (**ShareGen**) para procesar y luego repartir una parte $v_i \in \mathcal{V}_i$ del secreto s a cada participante P_i. Cuando un

subconjunto autorizado de participantes se reúne con sus partes, éstos pueden reconstruir el secreto s con sus partes a través de un algoritmo de recuperación del secreto (**Reconst**). El secreto s y ciertos aspectos del proceso de compartición son conocidos únicamente por el repartidor, mientras que el algoritmo de recuperación del secreto puede ser conocido por todos los participantes.

Usualmente se tiene que $\mathcal{V}_1 = \mathcal{V}_2 = \ldots = \mathcal{V}_n$, por lo que a dicho espacio de intercambio se le suele denotar por $\mathcal{V}$. Por lo general, los espacios $\mathcal{S}$ y $\mathcal{V}$ suelen ser campos finitos y/o espacios vectoriales.[1]

Definición 1.1.2. [36] Se denomina *conjunto de acceso* a un subconjunto de participantes $\hat{\mathcal{P}} \subseteq \mathcal{P}$ autorizado que puede reconstruir el secreto s con sus partes.

Definición 1.1.3. [36] Se denomina *conjunto mínimo de acceso* al conjunto de acceso en el que cualquier subconjunto propio de él no es un conjunto de acceso.

Definición 1.1.4. [36] Al conjunto de todos los conjuntos de acceso de un esquema de compartición de secretos se le denomina *estructura de acceso*.

Definición 1.1.5. [36] Se dice que un esquema de compartición de secretos tiene una *estructura de acceso monótona creciente* si cualquier superconjunto[2] de cualquier conjunto de acceso es también un conjunto de acceso.

En un esquema de compartición de secretos con estructura de acceso monótona creciente, dicha estructura se caracteriza totalmente por sus conjuntos mínimos de acceso.

1.2. Esquemas de compartición de secretos con umbral (k, n)

La característica principal de este tipo de esquemas es que sus conjuntos de acceso se definen por su tamaño como se describe a continuación:

Definición 1.2.1. [5] Sea $\mathcal{S}$ una estructura algebraica y $k, n \in \mathbb{Z}^+$ donde $k \leq n$. Entonces, un secreto $s \in \mathcal{S}$ se divide en n piezas o partes $v_1, \ldots, v_n$ de tal manera que:

1. El conocimiento de k o más partes v_i permite que s sea fácilmente calculado;

2. El conocimiento de $k-1$ o menos partes v_i deja a s completamente indeterminado (en el sentido de que todos sus valores posibles son igualmente probables).

A dicho esquema se le denomina *esquema de compartición de secretos con umbral* (k, n).

[1] Si se desea abundar sobre estas estructuras, sus propiedades, así como otras aplicaciones, se puede recurrir a los libros [37], [38] y [39].

[2] Se dice que un conjunto B es un superconjunto de un conjunto A si se cumple que $A \subseteq B$.

1.2.1. Esquema de compartición de secretos de A. Shamir

La idea principal de los esquemas que se presentan más adelante es elegir y reconstruir polinomios de grado $k-1$ con coeficientes en un campo. Una manera sencilla de reconstruir un polinomio es a través del método de interpolación de Lagrange. Este método permite obtener nuevos puntos partiendo de un conjunto discreto de puntos conocido como se describe a continuación:

Teorema 1.2.2. *[39] Sean $\alpha_0, \alpha_1, \ldots, \alpha_k$ elementos de un campo F y $\beta_0, \beta_1, \ldots, \beta_k$ elementos arbitrarios del mismo campo F, entonces existe un único polinomio:*

$$f(x) = \sum_{i=0}^{k} \beta_i \frac{(x-\alpha_0)\cdots(x-\alpha_{i-1})(x-\alpha_{i+1})\cdots(x-\alpha_k)}{(\alpha_i-\alpha_0)\cdots(\alpha_i-\alpha_{i-1})(\alpha_i-\alpha_{i+1})\cdots(\alpha_i-\alpha_k)}$$

de grado a lo más k tal que $f(\alpha_i) = \beta_i$ para $i = 0, 1, \ldots, k$. A $f(x)$ se le suele llamar polinomio interpolante de Lagrange.

Demostración. Ver en [39, p. 138]. □

La Figura 1-1 permite una mejor apreciación de lo descrito anteriormente para el campo $\mathbb{R}$. Los polinomios reconstruidos varían dependiendo de los puntos que se consideren. Nótese que la intersección de cada polinomio con el eje vertical es distinta a la de los demás.

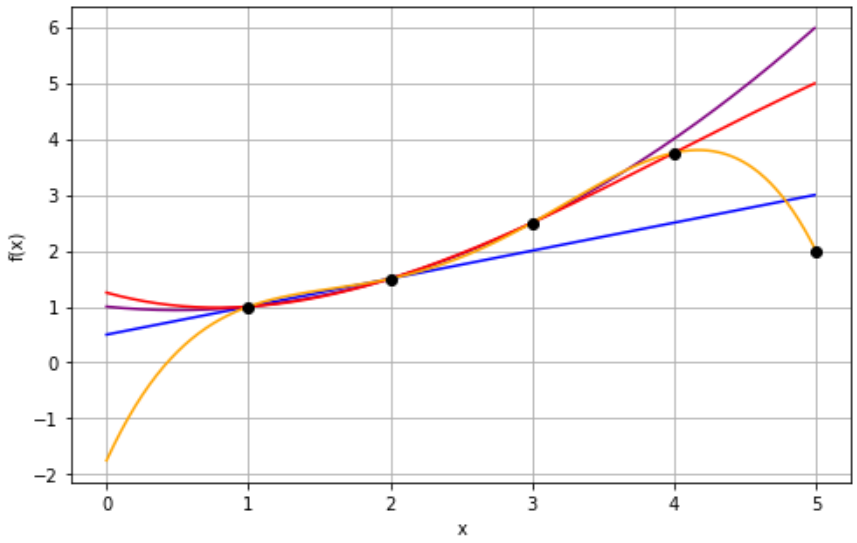

Figura 1-1: Ejemplo de polinomios interpolantes para algunos puntos dados.

Existen algoritmos[3] con una complejidad de $O(n\log^2 n)$ que permiten evaluar el polinomio interpolante de Lagrange. Sin embargo, una de las razones principales que convierte a éste método

[3] Ver [40, pág. 298]

una de las herramientas más utilizadas para los esquemas de compartición de secretos es su fácil implementación.

El primer esquema de compartición de secretos que se basó en este método es el esquema de compartición de secretos con umbral (k, n) de A. Shamir [5]. Este esquema permite compartir secretos que son elementos de un campo finito con un número primo p de elementos denotado por $\mathrm{GF}(p)$ [38], de hecho, la idea principal de este esquema es elegir y reconstruir polinomios de grado $k-1$ con coeficientes en dicho campo.

El esquema de A. Shamir puede generalizarse para compartir secretos que son elementos de un campo finito con un número q de elementos [38]. De acuerdo a esta generalización, los algoritmos **ShareGen** y **Reconst** se definen de la siguiente manera:

Algoritmo 1.2.1 ShareGen

Entrada: Secreto: $s \in \mathrm{GF}(q)$.
Salida: Lista of partes: $(v_1, \cdots, v_n)$.
1: Generar un polinomio aleatorio $f_s(x) \in \mathrm{GF}(q)[x]$ de grado $k-1$ tal que $f_s(0) = s$.
2: Calcular $v_i = f_s(i)$ y regresar $(v_1, \cdots, v_n)$.

Algoritmo 1.2.2 Reconst

Entrada: List of partes $(v_{i_1}, \cdots, v_{i_j})$ con $j \geq k$.
Salida: Secreto reconstruido $\hat{s}$.
1: Reconstruir un polinomio $\hat{f}_s(x)$ con $v_{i_1}, \cdots, v_{i_j}$ utilizando interpolación de Lagrange.
2: Regresar $\hat{s} = \hat{f}_s(0)$.

Ejemplo 1.2.3. Para brindar mayor claridad a los algoritmos anteriores considérese un esquema de compartición de secretos con umbral $(6, 3)$ y un secreto $s \in \mathrm{GF}(2^{12})$ cuya representación binaria es $s = 101101111001$.

De acuerdo al Algoritmo **ShareGen** se define un polinomio aleatorio de grado 2 con s como término constante, por ejemplo: $f(x) = 10010110100x^2 + 111110010010x + 101101111001$.

Luego se evalúa $f(x)$ en los identificadores de cada participante para generar sus partes:

$$\{(1, 10100110101), (110, 1110111010), (101, 10101101000),$$
$$(10, 11010001101), (100, 10011010101100111110), (11, 110110101011)\}$$

Para recuperar el secreto se utilizan las partes de un conjunto de acceso, el cual puede ser:

$$\{(1, 10100110101), (110, 1110111010), (101, 10101101000)\}$$

Y utilizando el método de interpolación de Lagrange se obtiene de nuevo el polinomio $f(x) = 110001000010x^2 + 1010110000x + 101101111001$. El cual, al evaluarse en 0, regresa el valor $\hat{s} = 101101111001$.

1.2.2. Propiedades

Algunas de las características más importantes que se busca en un esquema de compartición de secretos son su sencillez, su efectividad y escalabilidad. Los esquemas de compartición de secretos basados en el esquema de compartición de secretos con umbral (k, n) de A. Shamir [5] cuentan con dichas características.

Es muy importante tomar en cuenta la cantidad de información que se compartirá a los participantes del esquema, ya que lo deseable es compartir la menor cantidad de información posible, por ello es importante tener en cuenta lo siguiente:

"Un aspecto importante para la implementación de los esquemas de compartición de secretos es el tamaño de las partes, dado que la seguridad de un sistema se degrada de manera proporcional al aumento de la cantidad de información que debe mantenerse secreta. Lamentablemente en todos los esquemas de compartición de secretos el tamaño de las partes no puede ser menor al tamaño del secreto. De hecho, hay estructuras de acceso para las cuales el tamaño de las partes es estrictamente mayor al del secreto". [41, pág. 158]

Lo anterior se refiere a la cardinalidad de los espacios $\mathcal{V}$ y $\mathcal{S}$, es decir, $|\mathcal{V}|$ y $|\mathcal{S}|$. A estos valores suele llamárseles *el tamaño de las partes* y *el tamaño del secreto* respectivamente [1, 33, 34, 35].

Es fácil ver que de estos parámetros depende la cantidad de información que se desea proteger así como la cantidad de información que se deberá compartir. Además, estos dos parámetros permiten la comparación entre los esquemas a través de la relación que se describe a continuación:

Definición 1.2.4. [42] La *tasa de información* ρ de un esquema de compartición de secretos es la razón entre el tamaño del secreto y el tamaño de la parte más grande dada a algún participante, es decir $0 < \rho \leq 1$.

De acuerdo a [42] si el tamaño del secreto así como el de las partes se mide en bits y de acuerdo a la Definición 1.1.1 entonces se cumple lo siguiente:

$$\rho = \frac{\log_2|\mathcal{S}|}{\max \log_2|\mathcal{V}_i|}, \text{ con } i = 1, \ldots, n.$$

Definición 1.2.5. [41] Se dice que un esquema de compartición de secretos es *ideal* si $\rho = 1$.

1.3. Esquemas de compartición de secretos con detección de trampas

Como se mencionó anteriormente, uno de los escenarios que más se ha estudiado para los esquemas de compartición de secretos es el analizado por M. Tompa y H. Woll en [7], el cual se describe a continuación:

Se dice que uno o más participantes de un conjunto de acceso *hacen trampa* o *engañan* a los demás si modifican sus partes y proporcionan dichas partes alteradas al algoritmo **Reconst** de tal manera que el secreto reconstruido $\hat{s}$ sea válido pero no es el secreto compartido, es decir, $\hat{s} \in \mathcal{S}$ pero $\hat{s} \neq s$.

Debido a esta situación, la cual puede presentarse en la implementación de los esquemas de compartición de secretos, es necesario definir esquemas de compartición de secretos que permitan

detectar las trampas realizadas en el proceso de recuperación. Por lo tanto, es importante considerar lo siguiente:

Definición 1.3.1. [7] Un esquema de compartición de secretos es un *esquema de compartición de secretos con detección de trampas*, denotado *esquema de compartición de secretos* (k, n, ϵ)*-seguro* si:

1. Dicho esquema es un esquema de compartición de secretos con umbral (k, n).
2. Solo hay una pequeña probabilidad $\epsilon > 0$ de que cualesquiera $k-1$ participantes $p_1, p_2, \ldots, p_{k-1}$ puedan crear nuevas partes $v'_{i_1}, v'_{i_2}, \ldots, v'_{i_{k-1}}$ que puedan *engañar* a un k-ésimo participante p_k.

En el escenario descrito anteriormente, existen dos formas para modelar a un participante que desea hacer trampa dependiendo de lo que sabe: El modelo OKS (Ogata - Kurosawa - Stinson) [33], en el que un tramposo no sabe nada acerca del secreto ni de su distribución; y el modelo CDV (Carpentieri - De Santis - Vaccaro) [32], en el cual un tramposo puede saber lo que sea del secreto, incluso saber su contenido.

Si un esquema de compartición de secretos (k, n, ϵ)-seguro es seguro bajo el modelo OKS, entonces se denota como *esquema de compartición de secretos* (k, n, ϵ_{OKS})*-seguro*. Por otro lado, si es seguro bajo el modelo CDV, entonces se denota como *esquema de compartición de secretos* (k, n, ϵ_{CDV})*-seguro*.

Una forma bastante sencilla para detectar trampas durante la ejecución del algoritmo **Reconst** es a través de un *dígito verificador* $a \in \mathcal{S}$. Este dígito es el resultado de evaluar una *función verificadora* $A(x) : \mathcal{S} \to \mathcal{S}$ en el secreto s, es decir, $a = A(s)$. La función verificadora puede variar de un esquema a otro y cuando se detecta que algún participante hizo trampa, el algoritmo **Reconst** devuelve $\perp$.

1.3.1. Propiedades

Debido a que en estos esquemas se lleva a cabo un proceso adicional a la compartición de secretos (la detección de trampas), la información que se comparte a los participantes aumenta, es decir, la tasa de información ρ para estos esquemas siempre será menor a 1.

A pesar de esto, para este tipo de esquemas es importante estudiar el incremento mencionado anteriormente, así como la probabilidad ϵ de que un tramposo pueda engañar satisfactoriamente al resto de participantes en un conjunto de acceso. Para los esquemas de compartición de secretos (k, n, ϵ)-seguros que se presentarán en el resto del capítulo, ϵ dependerá de la definición de la función verificadora $A(x)$.

El primer análisis acerca del tamaño mínimo de la información que debe compartirse en este tipo de esquemas fue realizado en [7], desde entonces algunos autores han mejorado dicho análisis, las principales contribuciones fueron realizadas en [32] y [33]. Para los modelos OKS y CDV mencionados anteriormente, el tamaño de las partes, de acuerdo al máximo valor de ϵ, ha sido acotado inferiormente. En [33] se mostró que la cardinalidad de $\mathcal{V}$, $|\mathcal{V}|$, está acotada inferiormente bajo el modelo OKS como se describe a continuación:

$$|\mathcal{V}| \geq \frac{|\mathcal{S}| - 1}{\epsilon} + 1. \tag{1.3.1}$$

Y acotada inferiormente bajo el modelo CDV de la siguiente forma:

$$|\mathcal{V}| \geq \frac{|\mathcal{S}| - 1}{\epsilon^2} + 1. \tag{1.3.2}$$

1.4. Esquemas de compartición de secretos con detección eficiente de trampas

Dado que un secreto s es un elemento de una estructura algebraica $\mathcal{S}$, muchos tipos de secretos pueden ser *descompuestos* en algunos elementos de alguna estructura algebraica $\mathcal{T}$ con menos elementos que $\mathcal{S}$, es decir, $|\mathcal{T}| < |\mathcal{S}|$.

Propuestas recientes de esquemas de compartición de secretos con detección de trampas utilizan técnicas de descomposición en el secreto para evaluar una función verificadora $A(x) : \mathcal{S} \to \mathcal{T}$, lo cual permite obtener un dígito verificador $a \in \mathcal{T}$.

Ejemplo 1.4.1. Para ilustrar esto, retómese el Ejemplo 1.2.3. Una posible descomposición del secreto $s = 101101111001$ podría ser $s_1 = 1011$, $s_2 = 0111$ y $s_3 = 1001$. Como puede apreciarse, $s_1, s_2, s_3 \in \mathrm{GF}(2^4)$, por lo tanto, para una función $A(x) = x_1 \oplus x_2 \oplus x_3$, el dígito verificador tiene un valor de $a = A(s) = 0101$.

Para apreciar la detección de trampas a través del dígito verificador, se define un polinomio aleatorio $g(x)$ de grado 2 con a como término constante, por ejemplo: $g(x) = 1010x^2 + 1001x + 101$.

Luego se evalúa $f(x)$ en los identificadores de cada participante para generar sus partes:

$$\{(1, 111111000, 110), (110, 1110111010, 10111011),$$
$$(101, 10101101000, 10000010), (10, 11010001101, 111111),$$
$$(100, 10011010101100111110, 10000001), (11, 110110101011, 111100)\}$$

Luego se selecciona un conjunto de acceso, el cual puede ser:

$$\{(1, 111111000, 110), (110, 1110111010, 10111011), (101, 10101101000, 10000010)$$

Si ningún participante modifica el contenido de sus partes se obtienen los polinomios interpolantes $\hat{f}(x) = f(x)$ y $\hat{g}(x) = g(x)$. Al evaluarse dichos polinomios en 0 se obtienen los valores $\hat{s} = s$ y $\hat{a} = a$, finalmente, como $A(s) = a$, el esquema regresa el secreto s.

Sin embargo, supóngase que el participante con número identificador 1 reemplaza el segundo término de su parte por el valor 10000110.

Entonces, al evaluar el polinomio interpolante $\hat{f}(x)$ en 0 se obtiene el valor $\hat{s} = 100011011000$ y al evaluar $\hat{g}(x)$ en 0 se obtiene el valor $\hat{a}_1 = 0101$ el cual difiere de $A(\hat{s}) = 1000 \oplus 1101 \oplus 1000 = 1101$. Como consecuencia, el esquema detecta que el secreto reconstruido es erróneo y regresa $\perp$.

Considérese ahora el caso en que el participante con número identificador 1 puede hacer trampa de manera satisfactoria, por ejemplo, al reemplazar el segundo término de su parte por el valor 1011111.

Al evaluar los polinomios interpolantes $\hat{f}(x)$ y $\hat{g}(x)$ en 0 se obtienen los valores $\hat{s} = 111011110100$ y $\hat{a}_1 = 0101$ respectivamente, en este escenario $\hat{a}_1$ sí corresponde a $A(\hat{s}) = 1110 \oplus 1111 \oplus 0100 = 0101$ y por lo tanto el secreto erróneo $\hat{s}$ es regresado por el esquema.

El uso de técnicas de descomposición, como las descritas antes del Ejemplo 1.4.1, permite una detección de trampas más eficiente durante la implementación de **Reconst**. Además, el tamaño de las partes es menor al de los casos en que la función verificadora es de la forma $A(x) : \mathcal{S} \to \mathcal{S}$.

Algunos esquemas relevantes que utilizan técnicas de descomposición fueron presentados en [34], [35] y [1] y son los que se describen a continuación.

1.4.1. Esquema de T. Araki y W. Ogata

En 2011, T. Araki y W. Ogata propusieron un esquema en [34] que permite compartir secretos sobre $(\mathbb{Z}_p^*)^N$, es decir, vectores de dimensión N con entradas en el grupo multiplicativo $\mathbb{Z}_p^*$, donde p es un número primo. Nótese que los autores denotan por $\times$ al operador del grupo $\mathbb{Z}_p^*$.

Los algoritmos **ShareGen** y **Reconst** para este esquema se describen a continuación:

Algoritmo 1.4.1 ShareGen

Entrada: Secreto $s = (s_1, s_2, \ldots, s_N) \in (\mathbb{Z}_p^*)^N$ donde $N \geq 2$.
Salida: Lista de partes $(v_1, \cdots, v_n)$.

1: Generar N polinomios aleatorios $f_j(x) \in \mathrm{GF}(p)[x]$ de grado a lo más $k-1$, de tal forma que $f_j(0) = s_j$ para $1 \leq j \leq N$.
2: Calcular $c = s_1 \times \cdots \times s_N$ y generar un polinomio aleatorio $g(x) \in \mathrm{GF}(p)[x]$ de grado a lo más $k-1$ tal que $g(0) = c$.
3: Calcular $v_i = (f_1(i), \ldots, f_N(i), g(i))$ y regresar $(v_1, \cdots, v_n)$.

Algoritmo 1.4.2 Reconst

Entrada: Lista de partes $(v_{i_1}, \cdots, v_{i_k})$.
Salida: Secreto reconstruido $\hat{s}$ o $\perp$.

1: Reconstruir $\hat{s} = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N)$ y $\hat{c}$ con $v_{i_1}, \cdots, v_{i_k}$ usando interpolación de Lagrange, durante este proceso, si cualquier $\hat{s}_i = 0$ para $1 \leq j \leq N$ o bien $\hat{c} = 0$, regresar $\perp$ y detener la ejecución del algoritmo.
2: Regresar $\perp$ si $\hat{c} \neq \hat{s}_1 \times \cdots \times \hat{s}_N$. De otra forma regresar $\hat{s}$.

Para satisfacer la primera condición de la Definición 1.3.1, se puede hacer una pequeña mejora en los Pasos 1 y 2 de **ShareGen** (Algoritmo 1.4.1) al fijar el grado de $f_j(x)$ y $g(x)$ a $k-1$. No es difícil ver que, con esta mejora, el análisis hecho en [34] sigue siendo válido.

A pesar de que este esquema es eficiente y que con la mejora mencionada anteriormente es

seguro bajo el modelo OKS, dicho esquema no permite compartir de manera óptima todos los secretos que puedan ser expresados como cadenas de bits ya que excluye las entradas $s_i = 0$.

1.4.2. Esquemas de H. Hoshino y S. Obana

Otros esquemas de compartición de secretos relevantes, con detección eficiente de trampas son los que se describen a continuación, estos esquemas fueron presentados por H. Hoshino y S. Obana en 2015 [35] y 2016 [1], sin embargo, dado que el primer esquema es un caso particular del segundo, el análisis realizado aquí para [1] también es válido para [35].

Los algoritmos **ShareGen** y **Reconst** para [1] se definen de la siguiente manera:

Algoritmo 1.4.3 ShareGen

Entrada: Secreto $s \in \mathrm{GF}(2^{mN})$ donde $m \in \mathbb{Z}^+$ y N es un número entero par arbitrario.
Salida: Lista de partes $(v_1, \cdots, v_n)$.

1: Generar un polinomio aleatorio $f_s(x) \in \mathrm{GF}(2^{mN})[x]$ de grado $k-1$ de tal forma que $f_s(0) = s$.
2: Descomponer s para generar N cadenas de bits $s_1, s_2, \ldots, s_N \in \mathrm{GF}(2^m)$.
3: Generar el dígito verificador $a = s_1 \cdot s_2 + \cdots + s_{N-1} \cdot s_N$.
4: Generar un polinomio aleatorio $f_a(x) \in \mathrm{GF}(2^m)[x]$ de grado $k-1$ de tal forma que $f_a(0) = a$.
5: Calcular $v_i = (f_s(i), f_a(i))$ y regresar $(v_1, \cdots, v_n)$.

Algoritmo 1.4.4 Reconst

Entrada: Lista de partes $(v_{i_1}, \cdots, v_{i_j})$ donde $j \geq k$.
Salida: Secreto reconstruido $\hat{s}$ o $\perp$.

1: Reconstruir dos polinomios $\hat{f}_s(x)$ y $\hat{f}_a(x)$ con $v_{i_1}, \cdots, v_{i_j}$ usando interpolación de Lagrange.
2: Si se cumple que $\deg(\hat{f}_s) > k-1$ o que $\deg(\hat{f}_a) > k-1$, regresar $\perp$.
3: Calcular $\hat{s} = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N) = \hat{f}_s(0)$ y $\hat{a} = \hat{f}_a(0)$.
4: Regresar $\hat{s}$ si se cumple que $\hat{a} = \hat{s}_1 \cdot \hat{s}_2 + \cdots + \hat{s}_{N-1} \cdot \hat{s}_N$. De otra forma regresar $\perp$.

Se puede hacer una pequeña mejora en el Paso 2 de **Reconst** (Algoritmo 1.4.4) modificando dicho Paso de la siguiente manera: *"Si se cumple que* $\deg(\hat{f}_s) \neq k-1$ *o* $\deg(\hat{f}_a) \neq k-1$, *regresar* $\perp$*"*. Esta mejora se basa en el hecho de que en **ShareGen** (Algoritmo 1.4.3) el grado de los polinomios definidos es $k-1$, por lo que no es necesario continuar la ejecución del Algoritmo **Reconst** si los grados de los polinomios reconstruidos son menores a $k-1$, de hecho, si fuera el caso, las trampas y los intentos de que $k-1$ participantes intenten recuperar el secreto compartido se detectarían más rápido.

Para evaluar la probabilidad de trampa exitosa ϵ, H. Hoshino y S. Obana consideraron en [1, p. 626] la consistencia de la siguiente ecuación:

$$\hat{a} = \hat{s}_1 \cdot \hat{s}_2 + \cdots + \hat{s}_{N-1} \cdot \hat{s}_N. \tag{1.4.1}$$

Si se cumple dicha igualdad para $\hat{s}_i \neq s_i \; \forall \; 2 \leq i \leq N$ significa que un conjunto de tramposos $P_2, \cdots, P_k$ logró engañar a un participante honesto P_1.

Ahora, dado que el producto de dos elementos sobre un campo es cero si al menos uno de los elementos es cero, es más probable que dicho producto sea cero que un valor distinto de cero.

Por lo tanto, tomando en cuenta que (1.4.1) es una suma de productos de elementos sobre un campo, la probabilidad de trampa exitosa tiene dos valores posibles que corresponden a los casos en que el valor del dígito verificador es cero, denotado por $P(A(s) = 0)$, y un elemento distinto de cero $b \in \mathrm{GF}(2^m)^*$, denotado por $P(A(s) = b)$, además, a través de un análisis simple no es difícil ver que tales probabilidades satisfacen la siguiente relación:

$$P(A(s) = 0) > \frac{1}{2^m} > P(A(s) = b).$$

Esto quiere decir que $\epsilon = P(A(s) = 0)$ y por lo tanto, $\epsilon > 1/2^m$, lo cual difiere con lo establecido en [1, p. 626], en el sentido de que los autores concluyeron que $\epsilon = P(A(s) = c) = 1/2^m$ para cualquier $c \in \mathrm{GF}(2^m)$.

Debido a este comportamiento uno de los propósitos principales del presente trabajo es proponer un esquema de compartición de secretos (k, n, ϵ) seguro en el que se cumpla que $P(A(s) = c) = 1/2^m \ \forall \ c \in \mathrm{GF}(2^m)$.

Capítulo 2

Esquema propuesto

En el presente capítulo se define un esquema de compartición de secretos (k, n, ϵ_{OKS})-seguro. También se presenta una breve comparación de las propiedades de los esquemas en [34], [1] y las del esquema propuesto. Finalmente se incluyen los resultados de algunas pruebas realizadas a una implementación del esquema en [1] y del esquema propuesto.

2.1. Descripción

Los esquemas presentados en [34] y [1], los cuales se analizaron en el capítulo anterior, pueden ser modificados de manera sencilla para mejorar su comportamiento y propiedades. El esquema descrito a continuación está basado en dichos esquemas e incluye las mejoras propuestas en el Capítulo 1 para ellos.

El esquema propuesto permite compartir de manera óptima secretos $s \in \mathrm{GF}(2^{mN})$ donde $m, N \in \mathbb{Z}^+$ con $N > 1$. Además, en este esquema, a diferencia del esquema en [1], sí se cumple que $\epsilon = P(A(s) = c) = 1/2^m \ \forall \ c \in \mathrm{GF}(2^m)$.

Los algoritmos **ShareGen** y **Reconst** del esquema propuesto se definen de la siguiente manera:

Algoritmo 2.1.1 ShareGen

Entrada: Secreto $s \in \mathrm{GF}(2^{mN})$ donde $m, N \in \mathbb{Z}^+$ con $N > 1$.
Salida: Lista de partes $(v_1, \cdots, v_n)$.

1: Generar un polinomio aleatorio $f_s(x) \in \mathrm{GF}(2^{mN})[x]$ de grado $k-1$ de tal forma que $f_s(0) = s$.
2: Descomponer a s para generar N cadenas de bits $s_1, s_2, \ldots, s_N \in \mathbb{Z}_{2^m}$.
3: Calcular $a = s_1 + s_2 + \cdots + s_N \pmod{2^m}$.
4: Generar un polinomio aleatorio $f_a(x) \in \mathrm{GF}(2^m)[x]$ de grado $k-1$ de tal forma que $f_a(0) = a$.
5: Calcular $v_i = (f_s(i), f_a(i))$ y regresar $(v_1, \cdots, v_n)$.

Algoritmo 2.1.2 Reconst

Entrada: Lista de partes $(v_{i_1}, \cdots, v_{i_j})$ donde $j \geq k$.
Salida: Secreto reconstruido $\hat{s}$ o $\perp$.

1: Reconstruir dos polinomios $\hat{f}_s(x)$ y $\hat{f}_a(x)$ con $v_{i_1}, \cdots, v_{i_j}$ utilizando interpolación de Lagrange.
2: Si se cumple que $\deg(\hat{f}_s) \neq k-1$ o que $\deg(\hat{f}_a) \neq k-1$, entonces regresar $\perp$.
3: Calcular $\hat{s} = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N) = \hat{f}_s(0)$ y $\hat{a} = \hat{f}_a(0)$
4: Regresar $\hat{s}$ si se cumple que $\hat{a} = \hat{s}_1 + \hat{s}_2 + \cdots + \hat{s}_N \pmod{2^m}$. De otra forma regresar $\perp$.

De manera similar a cómo lo hicieron H. Hoshino y S. Obana, las principales propiedades de nuestro esquema se pueden resumir a través del siguiente teorema:

Teorema 2.1.1. *Si el secreto s se distribuye uniformemente sobre GF(2^{mN}), entonces el esquema propuesto es un esquema de compartición de secretos (k, n, ϵ_{OKS})-seguro. Las principales propiedades del esquema son $|\mathcal{S}| = 2^{mN}$, una probabilidad de trampa exitosa $\epsilon = 1/2^m$ y $|\mathcal{V}| = |\mathcal{S}|/\epsilon = 2^{N(m+1)}$. Además, el tamaño de las partes alcanza la cota inferior en (1.3.1).*

Demostración. De acuerdo a la descripción del esquema, sean $\mathcal{A} = \mathbb{Z}_{2^m}$ y $\mathcal{S} = \text{GF}(2^{mN})$, entonces $|\mathcal{A}| = 2^m$ y $|\mathcal{S}| = 2^{mN}$.

Ahora, como se describe en el Algoritmo 2.1.1, $v_i = (f_s(i), f_a(i))$ donde $v_i \in \mathcal{V}$, $f_s(i) \in \mathcal{S}$ y $f_a(i) \in \mathcal{A}$, entonces:

$$|\mathcal{V}| = |\mathcal{S}| \cdot |\mathcal{A}| = 2^{mN} \cdot 2^m = 2^{m(N+1)}.$$

Ahora, para determinar el valor de ϵ, considérese la siguiente ecuación:

$$\hat{a} = (\hat{s}_1 + \hat{s}_2 + \cdots + \hat{s}_N) \pmod{2^m}. \tag{2.1.1}$$

Si ningún participante hace trampa, entonces durante la ejecución del Algoritmo **Reconst** se satisface lo siguiente:

$$s_1 = \hat{s}_1,\ s_2 = \hat{s}_2, \ldots,\ s_N = \hat{s}_N,\ a = \hat{a}.$$

Para que un participante pueda hacer trampa de manera satisfactoria, debe cumplirse en (2.1.1) que al menos un $\hat{s}_i \neq s_i$.

Como $\hat{a}, \hat{s}_1, \ldots, \hat{s}_N \in \mathbb{Z}_{2^m}$ y se operan mediante la suma en dicho anillo, los posibles valores de $\hat{a}$ en (2.1.1) se distribuyen de manera uniforme. Entonces la probabilidad de que $\hat{a}$ sea algún elemento $b \in \mathbb{Z}_{2^m}$ para valores aleatorios de $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N$ es:

$$P(\hat{a} = b) = \frac{1}{2^m}. \tag{2.1.2}$$

Por lo tanto la probabilidad de trampa exitosa para el esquema propuesto es $\epsilon = 1/2^m$.

Nótese que la siguiente relación se cumple para los valores determinados de ϵ y de $|\mathcal{V}|$:

$$|\mathcal{V}| = \frac{2^{mN}}{\frac{1}{2^m}} = \frac{|\mathcal{S}|}{\epsilon}.$$

Ahora, considérese una parte $v_i \in \mathcal{V}$ con tamaño $|\mathcal{V}|$, entonces la longitud en bits de la parte está dada por:

$$\lceil \log_2(|\mathcal{V}|) \rceil = m(N+1).$$

Como se describe en (1.3.1), la cota inferior del tamaño de las partes en un esquema de compartición de secretos (k, n, ϵ_{OKS})-seguro, denotada aquí por $|\mathcal{V}_{OKS}|$, es:

$$|\mathcal{V}_{OKS}| = \frac{|\mathcal{S}| - 1}{\epsilon} + 1.$$

Entonces, la longitud en bits de la cota $|\mathcal{V}_{OKS}|$ debe satisfacer lo siguiente:

$$\lceil \log_2(|\mathcal{V}_{OKS}|) \rceil = \lceil \log_2(\frac{|\mathcal{S}| - 1}{\epsilon} + 1) \rceil.$$

Dado un secreto $s \in \mathcal{S}$ con tamaño $|\mathcal{S}| = 2^{mN}$ entonces:

$$\begin{aligned}
\lceil \log_2(|\mathcal{V}_{OKS}|) \rceil &= \lceil \log_2(\frac{2^{mN} - 1}{2^{-m}} + 1) \rceil \\
&= \lceil \log_2(2^{m(N+1)} - 2^m + 1) \rceil \\
&= \lceil \log_2(2^{m(N+1)}(1 - \frac{2^m}{2^{m(N+1)}} + \frac{1}{2^{m(N+1)}})) \\
&= \lceil \log_2(2^{m(N+1)}(1 - \frac{1}{2^{mN}} + \frac{1}{2^{m(N+1)}})) \rceil \\
&= \lceil \log_2(2^{m(N+1)}) + \log_2(1 - \frac{1}{2^{mN}} + \frac{1}{2^{m(N+1)}}) \rceil \\
&= \lceil m(N+1) + \log_2(1 - \frac{1}{2^{mN}} + \frac{1}{2^{m(N+1)}}) \rceil.
\end{aligned}$$

Para la última ecuación, el segundo término satisface lo siguiente:

$$-1 < \log_2(1 - \frac{1}{2^{mN}} + \frac{1}{2^{m(N+1)}}) < 0.$$

Reescribiendo de otra manera se tiene lo siguiente:

$$\frac{1}{2^{mN}} - \frac{1}{2} < \frac{1}{2^{m(N+1)}} < \frac{1}{2^{mN}}.$$

En donde las desigualdades se cumplen para $m, N \in \mathbb{Z}^+$. Por lo tanto, se tiene que:

$$\lceil \log_2(|\mathcal{V}_{OKS}|) \rceil = m(N+1).$$

Lo cual quiere decir que para el esquema propuesto se cumple que:

$$\lceil \log_2(|\mathcal{V}|) \rceil = \lceil \log_2(|\mathcal{V}_{OKS}|) \rceil.$$

Es decir, el tamaño de las partes del esquema propuesto alcanza la cota inferior en (1.3.1). □

2.2. Comparación de propiedades

El esquema propuesto, de manera similar al presentado por H. Hoshino y S. Obana en [1], admite cualquier secreto que pueda expresarse como una cadena de bits, además, como puede apreciarse en el Cuadro 2.1, la única restricción para N es que tiene que ser un entero positivo mayor a 1, por lo que el esquema propuesto puede manejar de manera óptima más secretos que el esquema en [1]. Además, a diferencia de los esquemas en [34] y [1], el tamaño de las partes en el esquema propuesto sí alcanza la cota inferior en (1.3.1).

Cuadro 2.1: Propiedades de los esquemas analizados y del esquema propuesto.

Esquema	Esquema propuesto	T. Araki y W. Ogata [34]	H. Hoshino y S. Obana [1]
Complejidad de los algoritmos	$\mathcal{O}(n\log^2 n)$	$\mathcal{O}(n\log^2 n)$	$\mathcal{O}(n\log^2 n)$
Tamaño de las partes	$\|\mathcal{V}\| = \|\mathcal{S}\|/\epsilon$	$\|\mathcal{V}\| = p^{N+1} \approx \|\mathcal{S}\|/\epsilon$	$\|\mathcal{V}\| > \|\mathcal{S}\|/\epsilon$
Secretos soportados	$s \in \mathrm{GF}(2^{mN})$ con $N > 1$	$s \in (\mathbb{Z}_p^*)^N$	$s \in \mathrm{GF}(2^{mN})$ con N par
Evaluación de la función verificadora	sobre $\mathbb{Z}_{2^m}$	sobre $\mathbb{Z}_p^*$	sobre $\mathrm{GF}(2^m)$
Complejidad de la función verificadora	$\mathcal{O}(n)$	$\mathcal{O}(n\log^2 n)$	$\mathcal{O}(n\log^2 n)$

La evaluación de la función verificadora del esquema propuesto es eficiente, pues aplica una técnica de descomposición sobre el secreto. Además, como la mayoría de las veces es un poco más rápido y sencillo operar cadenas de bits en una computadora como elementos de $\mathbb{Z}_n$ que operarlos como elementos de un campo finito, la implementación de la función verificadora es más sencilla y su evaluación más rápida. Lo anterior es claramente una ventaja más respecto al esquema en [1].

Para ilustrar lo anterior, las funciones verificadoras del esquema en [1] y del esquema propuesto fueron implementadas en Python (véase el código en la Sección A.1).

Las gráficas de las Figuras 2-1 y 2-2 muestran el tiempo promedio transcurrido para calcular el dígito verificador a a partir de secretos $s \in \mathrm{GF}(2^{mN})$ generados pseudo-aleatoriamente para algunos valores de m y N. Cada nodo de las gráficas es el tiempo promedio de $10,000$ dígitos verificadores calculados.

Nótese que los valores de N utilizados son pares, esto permite que ambos esquemas puedan compartir de manera óptima los secretos generados. En cuanto a los valores de m, se escogieron valores pequeños y suficientemente alejados entre sí para que se puedan apreciar los aumentos en los tiempos de evaluación de las funciones verificadoras.

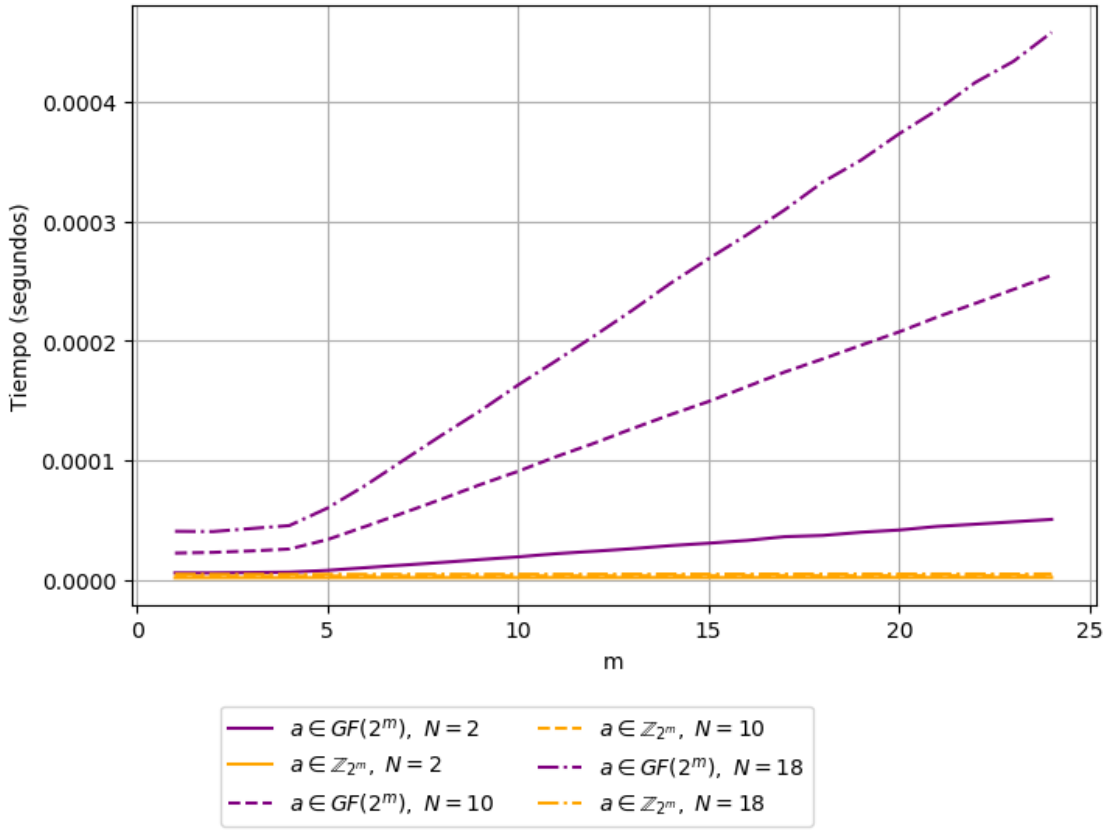

Figura 2-1: Tiempo promedio requerido para evaluar la función verificadora del esquema propuesto en [1] (curvas púrpuras) y la función verificadora propuesta en el presente trabajo (curvas naranjas).

Como puede apreciarse en la Figura 2-1, el tiempo transcurrido para la evaluación de la función verificadora de H. Hoshino and S. Obana (las tres curvas púrpuras) crece de manera proporcional a N y a m, además sus valores varían entre 5×10^{-06} [seg] y 4.8×10^{-04} [seg].

Por otro lado, las evaluaciones de la función verificadora propuesta (curvas naranjas que apa-

recen en la parte inferior de la Figura 2-1) se mantienen casi constantes. De hecho, como puede apreciarse en la Figura 2-2, el tiempo transcurrido para la evaluación de la función verificadora propuesta se mantiene entre 1.3×10^{-06} [seg] y 4.5×10^{-06} [seg] y oscila cerca de algunos valores constantes que básicamente dependen de N.

Esto significa que, para una probabilidad ϵ fija y para secretos más grandes, el tiempo transcurrido para la evaluación de la función verificadora aumenta para el esquema en [1] y permanece casi constante para el esquema propuesto.

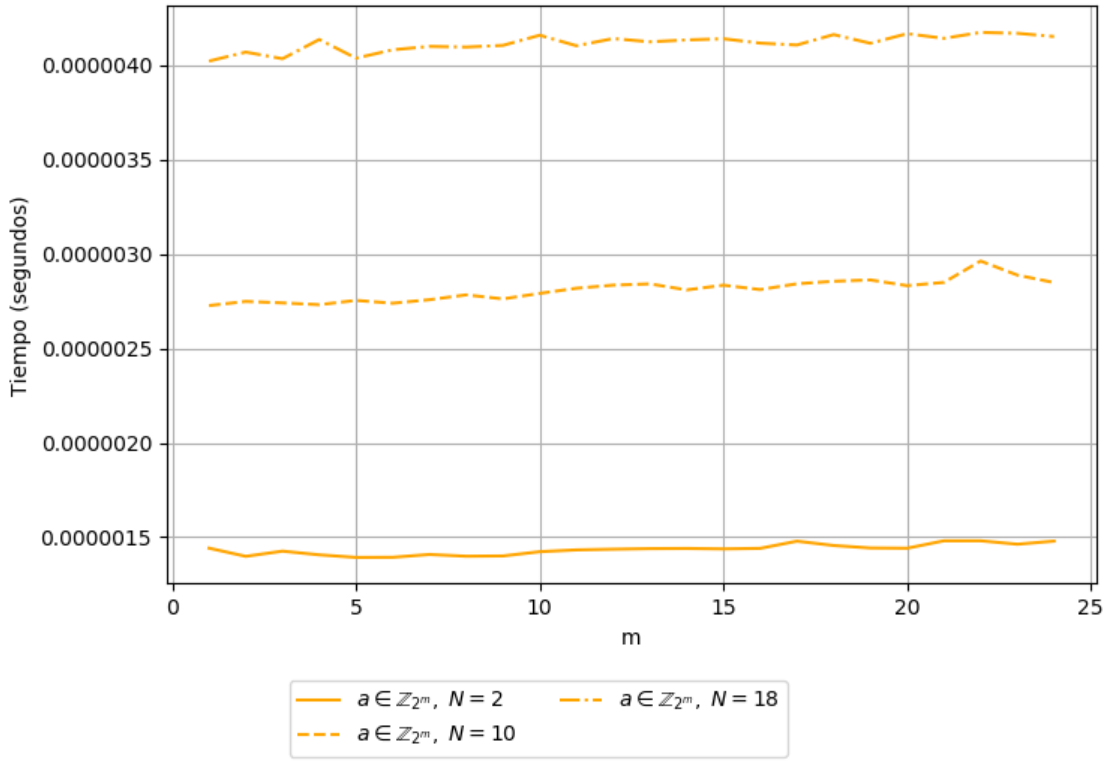

Figura 2-2: Tiempo promedio requerido para evaluar la función verificadora propuesta.

2.3. Aplicación a la protección de datos sensibles

Para complementar la comparación que se realizó previamente, tanto el esquema en [1] como el esquema propuesto se implementaron en Python (véase código en la Sección A.2) y se aplicaron a un caso muy básico de resguardo de datos sensibles.

El escenario consiste en un servidor principal cuya información se desea resguardar en seis servidores de respaldo de tal manera que se requiera de al menos tres de ellos para recuperar la información respaldada, esta información se respaldará en bloques de 16 bits, por lo que cada bloque puede ser manejado como un elemento de $\mathrm{GF}(2^{16})$.

Para realizar una comparación más completa del comportamiento de ambos esquemas, se rea-

lizaron tres pruebas para el proceso de recuperación del secreto:

1. Intento de recuperar los datos resguardados con un conjunto de acceso de tamaño válido y con las partes sin modificar,

2. Intento de recuperar los datos resguardados con un conjunto de acceso de tamaño menor al umbral, y

3. Intento de recuperar los datos con al menos una parte modificada en el conjunto de acceso.

Los resultados de este experimento se resumen en la Figura 2-3 para $10,000$ bloques de datos. Las dos primeras columnas en dicho gráfico, corresponden a la recuperación exitosa de datos. Por otro lado, la recuperación de un secreto incorrecto o la detección de un conjunto de acceso no válido se representan a través de la tercera y cuarta columnas, finalmente los resultados para la realización de una trampa de manera no exitosa o la salida de error de los esquemas, aparecen en las dos últimas columnas.

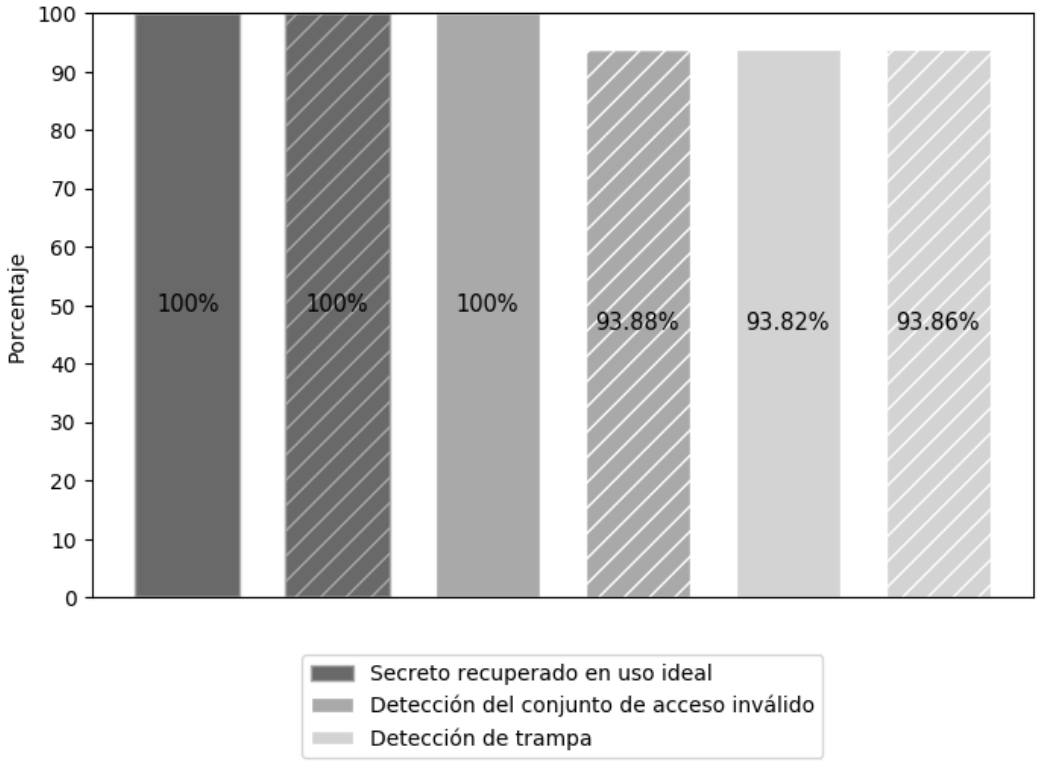

Figura 2-3: Comparación del desempeño del esquema de H. Hoshino y S. Obana (ashurado) vs. el esquema propuesto (color plano).

Como puede verse en la Figura 2-3, el rendimiento de ambos esquemas en la primera prueba fue 100 % exitoso, esto significa que se recuperaron todos los datos protegidos.

En la segunda prueba, el esquema propuesto detectó todos los intentos de recuperar los datos resguardados a través de conjuntos de acceso no válidos, en contraposición al esquema en [1] en el que sólo se detectaron el 93.88 % de los conjuntos de acceso no válidos, y los datos recuperados por el otro 6.12 % de intentos fueron erróneos. Finalmente, la tercera prueba corrobora el comportamiento reportado para ambos esquemas en la Subsección 1.4.2 y la Sección 2.1 respectivamente.

Conclusiones

Se propuso un esquema de compartición de secretos, el cual, como se describe en el Teorema 2.1.1, es un esquema de compartición de secretos (k, n, ϵ_{OKS})-seguro, es eficiente y el tamaño de las partes alcanza la cota inferior en (1.3.1). De hecho, las principales propiedades del esquema propuesto son las siguientes:

$$|\mathcal{S}| = 2^{mN},\ \epsilon = 1/2^m,\ |\mathcal{V}| = 2^{m(N+1)}$$

El esquema propuesto satisface las propiedades reportadas para el esquema en [1] (como se puede apreciar en el Cuadro 2.1). Además permite compartir de manera optima más secretos que puedan ser representados por cadenas de bits de longitud mN con N no necesariamente par, pero sí mayor a 1.

La implementación del esquema propuesto requiere un menor costo computacional para evaluar la función verificadora que en las funciones de los esquemas en [34] y [1]. Además, dicha función es más fácil de implementar.

A pesar de que el comportamiento del esquema propuesto es mejor que el del esquema en [1], el reto abierto en [1] acerca de la construcción de un esquema que satisfaga los siguientes puntos permanece abierto:

1. El esquema debe garantizar seguridad bajo el modelo OKS cuando el secreto es un elemento de GF(2^m).
2. El costo computacional del esquema debe ser muy bajo.
3. La estructura algebraica para la implementación del esquema no debe depender de la estructura sobre la que se distribuye el secreto.

En otras palabras: dicho esquema debe ser un esquema de compartición de secretos (k, n, ϵ_{OKS})-seguro que permita compartir de manera óptima secretos $s \in$ GF(2^m) para cualquier valor de m, además debe ser eficiente y el tamaño de la estructura algebraica sobre la que se realicen las operaciones de la función verificadora no debe depender del valor de m.

Apéndice A

Código fuente de los programas realizados

Los programas presentados a continuación fueron realizados para ilustrar el funcionamiento del esquema propuesto versus el del esquema de H. Hoshino y S. Obana [1]. Es importante tomar en cuenta que el tiempo de ejecución de estos programas se verá afectado por el tamaño del campo finito sobre el cual se defina el secreto a compartir.

El primer programa sólo permite construir campos finitos GF(2^m) con $m < 25$, debido a esto el segundo programa sólo puede compartir secretos con una longitud de 24 bits o menor. Sin embargo, si se desea compartir un secreto con una longitud mayor, entonces puede dividirse dicho secreto en bloques con una longitud menor a 25 bits y compartir cada bloque como si fuese un secreto independiente. De esta forma se reduce el tiempo de ejecución de los programas y se mantienen las propiedades de los esquemas como si se aplicaran directamente al secreto completo. Es decir, la probabilidad de hacer trampa satisfactoriamente es bastante reducida y por lo tanto la seguridad del proceso de compartición del secreto es alta.

A.1. Código fuente de la prueba realizada a la evaluación de las funciones verificadoras del esquema en [1] y el esquema propuesto

El programa *CBinario.py*, cuyo código fuente se presenta a continuación, fue realizado en Python 3 y para ser ejecutado requiere los paquetes *numpy*, *time*, *random* y *matplotlib* de Python.

Para correr el programa ejecute la siguiente línea en la consola:

python3 CBinario.py

Código A.1: CBinario.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Mar  4 19:58:44 2018
```

```
4
5 @author: Daniel Becerra Pedraza
6
7 Este programa permite construir un campo de caracteristica 2 con 2^{m}
8 elementos con m<25, si se ejecuta este archivo entonces se muestran unas
9 pruebas basicas, asi como los resultados de unas pruebas para la evaluacion
10 de las funciones verificadoras de los esquemas de comparticion de secretos
11 propuestos por H. Hoshino y S. Obana (2016) y D. Becerra y G. Vega (2019).
12
13 """
14 import numpy as np
15
16 #operaciones utilizadas para elementos de GF(2^m):
17 class GF2():
18
19     def __init__(self, m = None):
20         """
21         Constructor de la clase.
22         """
23         self.__m = int(m)
24         self.__q = int(2**m)
25         paux = np.zeros(24)
26         paux[0] = 0x3
27         paux[1] = 0x7
28         paux[2] = 0xb
29         paux[3] = 0x13
30         paux[4] = 0x25
31         paux[5] = 0x43
32         paux[6] = 0x89
33         paux[7] = 0x11b
34         paux[8] = 0x211
35         paux[9] = 0x409
36         paux[10] = 0x805
37         paux[11] = 0x1009
38         paux[12] = 0x201b
39         paux[13] = 0x4443
40         paux[14] = 0x8003
41         paux[15] = 0x1002b
42         paux[16] = 0x20009
43         paux[17] = 0x40081
44         paux[18] = 0x80027
45         paux[19] = 0x100009
46         paux[20] = 0x200005
47         paux[21] = 0x400003
48         paux[22] = 0x800021
49         paux[23] = 0x1000087
50         self.__pirr =  int(paux[m-1])
51
52     def __del__(self):
53         """
54         Destructor de la clase.
55         """
56         del(self.__q)
57         del(self.__pirr)
58
59     def add(self,a,b):              #Suma
```

```
60          """
61          Esta funcion realiza la operacion xor entre dos valores a y b.
62
63          @param a: elemento del campo [valor entero].
64          @param b: elemento del campo [valor entero].
65
66          @return a: a xor b.
67          """
68          return a^b
69
70      def mult(self,a,b):           #multiplicacion
71          """
72          Esta funcion realiza la multiplicacion de dos valores enteros
73          definida para el campo GF(2^m).
74
75          @param a: elemento del campo [valor entero].
76          @param b: elemento del campo [valor entero].
77
78          @return a: a . b mod (self.__pirr).
79          """
80          i=0
81          r=0
82          while True:
83              if (int(b)>>i)&1:
84                  r=r^(int(a)<<i)
85              if (int(b)>>(i+1))==0:
86                  break
87              i+=1
88          a=r
89          ncp=np.ceil(np.log2(self.__pirr+1))
90          while True:
91              nca=np.ceil(np.log2(a+1))
92              if nca < ncp:
93                  break
94              a = a ^ (self.__pirr<<int(nca-ncp))
95          return a
96
97      def pot(self,a,n):            #potencia
98          """
99          Esta funcion multiplica un valor entero consigo mismo n veces
100         con la multiplicacion definida para el campo GF(2^m).
101
102         @param a: elemento del campo [valor entero].
103         @param n: exponente [valor entero].
104
105         @return a: a^n.
106         """
107         if n == 0:
108             return 0x1
109         b = a
110         for i in range(0,n-1):
111             b = self.mult(b,a)
112         return b
113
114     def inv(self,a):
115         """
```

```
116         Esta funcion regresa el inverso multiplicativo de un elemento del campo GF(2^m
                ) utilizando el Algoritmo Extendido de Euclides para encontrarlo.
117
118         @param a: elemento del campo [valor entero].
119
120         @return a: a^{-1}.
121         """
122         r0, r1 = a, self.__pirr
123         s0, s1 = 1, 0
124         while True:
125             d, b = r0, r1
126             ncb=np.ceil(np.log2(b+1))
127             c=0
128             while True:
129                 ncd=np.ceil(np.log2(d+1))
130                 if ncd < ncb:
131                     break
132                 c = c^1<<int(ncd-ncb)
133                 d = d ^ (b<<int(ncd-ncb))
134             if not d:
135                 break
136             r1, r0 = d, r1
137             i=0
138             r=0
139             while True:
140                 if (int(c)>>i)&1:
141                     r=r^(int(s1)<<i)
142                 if (int(c)>>(i+1))==0:
143                     break
144                 i+=1
145             s = s0^r
146             s1, s0 = s, s1
147         return s1
148
149     def m(self):
150         return self.__m
151
152
153     def q(self):
154         return self.__q
155
156     def pirr(self):
157         return self.__pirr
158
159
160 if __name__ == '__main__':
161
162     import time
163     import random as rnd
164     import matplotlib.pyplot as plt
165     ############################################################
166     #Prueba 1: Operaciones sobre GF
167     ############################################################
168     m=16
169     a=2
170     b=6
```

```
171     campo2 = GF2(m)
172     s = campo2.add(a,b)
173     print("s = " + str(s))
174     m = campo2.mult(a,b)
175     print("m = " + str(m))
176     i = campo2.inv(a)
177     print("i = " + str(i))
178
179     ########################################################
180     #Prueba 2: Calculo de dos digitos verificadores
181     ########################################################
182     #se definen los parametros del esquema
183     k = 3
184     m = 5
185     N = 6
186     q2 = 2**m
187     campo2 = GF2(m)
188     Nmin=1
189     Nmax=12
190     mmin=0
191     mmax=23
192     nej=10000     #numero de ejecuciones
193
194     plt.figure(figsize=(8,5))
195     plt.style.use('default')
196     SOt = np.zeros((Nmax+1,mmax+1))
197     Pt = np.zeros((Nmax+1,mmax+1))
198     x = np.linspace(1,mmax+1,mmax-mmin+1)
199     ptrn=("-","--","-.",":","---","----","---",":","-.","-.")
200     for j in range(Nmin,Nmax+1,4):
201         N=2*j
202         for k in range(mmin,mmax+1):
203             q=2**k
204             s = np.zeros(N)
205             for l in range(0,nej):
206                 for i in range(0,N):
207                     s[i]=rnd.randint(0,q-1)
208                 sumaaux = 0
209                 t1 = time.clock()
210                 for i in range(0,j):
211                     sumaaux =campo2.add(sumaaux,campo2.mult(s[2*i],s[2*i+1]))
212                 t2 = time.clock()
213                 SOt[j,k] += (t2 - t1)/nej
214                 sumaaux = 0
215                 t1 = time.clock()
216                 for i in range(0,N):
217                     sumaaux =sumaaux+s[i]
218                 sumaaux=sumaaux%q2
219                 t2 = time.clock()
220                 Pt[j,k] += (t2 - t1)/nej
221         plt.plot(x,SOt[j],ptrn[j-1], color='purple', label='$a \in GF(2^m),\ N=' + str
                (N) + '$')
222         plt.plot(x,Pt[j],ptrn[j-1], color='orange',label='$a \in \mathbb{Z}_{2^{m}},\
                N=' + str(N) + '$')
223     plt.legend(loc='upper center', bbox_to_anchor=(0.4, -0.15),  shadow=False, ncol=2)
224     plt.xlabel('m')
```

```
225     plt.ylabel('Tiempo (segundos)')
226     plt.grid()
227     plt.show()
```

A.2. Código de las pruebas realizadas a la implementación del esquema en [1] y del esquema propuesto

El programa *Prototipo.py*, cuyo código fuente se presenta a continuación, fue realizado en Python 3 y para ser ejecutado requiere los paquetes *numpy*, *random*, *matplotlib* y *os* de Python así como la clase *GF*2 del programa *CBinario.py*.

Para correr el programa ejecute la siguiente línea en la consola:

python3 Prototipo.py

Código A.2: Prototipo.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Mar  4 19:58:44 2018
4
5  @author: Daniel Becerra Pedraza
6
7  Este programa consiste en una implementacion de los esquemas de comparticion de
8  secretos propuestos por H. Hoshino y S. Obana (2016) y D. Becerra y G. Vega
9  (2019).
10
11 Para ambos esquemas se llevan a cabo 3 pruebas:
12
13 1. Proceso ideal de recuperacion del secreto. (CAV)
14 2. Intento de recuperar el secreto con un conjunto de acceso no valido. (CANV)
15 3. Intento de recuperar el secreto con al menos una parte modificada. (T)
16
17 """
18 import numpy as np
19 import random as rnd
20 import matplotlib.pyplot as plt
21 from CBinario import GF2
22 import os
23
24 def elementoAleatorio(q):
25     """
26     Esta funcion regresa un entero 'aleatorio' de un campo dado <campo>
27     con <q> elementos.
28
29     @param q: numero de elementos que tiene el campo.
30
31     @return: elemento aleatorio del campo <campo>.
32     """
33     os.urandom
```

```
34      return int(rnd.randint(0,q-1))
35
36  def polinomioAleatorio(grado, termcte, q):
37      """
38      Esta funcion genera un polinomio 'aleatorio' de grado <grado>
39      cuyos coeficientes son elementos de un campo dado <campo> con <q>
40      elementos y cuyo termino constante es el elemento <termcte>.
41
42      @param grado: grado del polinomio a generar.
43      @param termcte: valor del termino constante del polinomio a generar.
44      @param q: numero de elementos que tiene el campo.
45
46      @return P: vector con los coeficientes del polinomio generado.
47      """
48      P = []
49      P.append(termcte)
50      for i in range(0,grado):
51          P.append(elementoAleatorio(q))
52      return P
53
54  def evaluaPolinomio(P, val, campo):
55      """
56      Esta funcion evalua un polinomio <P> en determinado valor <val>.
57
58      @param P: vector con los coeficientes del polinomio
59      @param val: valor en el cual se evaluara el polinomio
60
61      @return Px: valor del polinomio <P> evaluado en <val>
62      """
63      Px = P[0]
64      xi=1
65      for i in range(1,len(P)):
66          xi = campo.mult(xi,val)
67          Px = campo.add(Px,campo.mult(xi,P[i]))
68      return Px
69
70
71  def construyeValorVerificador(campo, secreto, m, N, Prop):
72      """
73      Esta funcion construye el valor verificador del secreto <secreto>
74      de acuerdo a lo seleccionado en el parametro <Prop>.
75
76      @param campo: campo sobre el que se encuentra el valor verificador y las
77          segmentaciones del secreto en el esquema de H. Hoshino y S. Obana.
78      @param secreto: secreto que se va a segmentar.
79      @param m: numero de bits de los segmentos del secreto.
80      @param N: numero de segmentos.
81      @param Prop: 1 para aplicar el esquema de Becerra y Vega, cualquier otro
82          entero para aplicar el esquema de Hoshino y Obana.
83
84      @return a: valor verificador construido
85      """
86      if Prop == 1:
87          return construyeValorVerificadorBV(secreto, m, N)
88      else:
89          return construyeValorVerificadorHO(campo, secreto, m, N)
```

```
90
91  def construyeValorVerificadorBV(secreto, m, N):
92      """
93      Esta funcion construye el valor verificador del secreto <secreto>
94      de acuerdo a lo propuesto por Daniel Becerra y Gerado Vega:
95
96          a = (s_{1}+s_{2}+s_{3}...+s_{N})mod(2^{m})
97
98      @param secreto: secreto que se va a segmentar.
99      @param m: numero de bits de los segmentos del secreto.
100     @param N: numero de segmentos.
101
102     @return a: valor verificador construido.
103     """
104     q=2**m
105     bitstr = bin(secreto)
106     bitstr = bitstr[2:]
107     x = len(bitstr)
108     aaux = 0
109     while x<N*m:
110         bitstr = "0" + bitstr
111         x +=1
112     for i in range(0,N):
113         aaux = aaux + int(bitstr[i*m:(i+1)*m],2)
114     a = int(aaux%q)
115     return a
116
117 def construyeValorVerificadorHO(campo, secreto, m, N):
118     """
119     Esta funcion construye el valor verificador del secreto <secreto>
120     de acuerdo a lo propuesto por Hidetaka Hoshino y Satoshi Obana:
121
122         a = s_{1}*s_{2}+s_{3}*s_{4}+...+s_{N-1}*s_{N}
123
124     @param campo: campo sobre el que se encuentra el valor verificador
125         y las segmentaciones del secreto.
126     @param secreto: secreto que se va a segmentar.
127     @param m: numero de bits de los segmentos del secreto.
128     @param N: numero de segmentos.
129
130     @return a: valor verificador construido.
131     """
132     bitstr = bin(secreto)
133     bitstr = bitstr[2:]
134     x = len(bitstr)
135     a = 0
136     while x<N*m:
137         bitstr = "0" + bitstr
138         x +=1
139     for i in range(0,int(N/2)):
140         a = campo.add(a,campo.mult(int(bitstr[(2*i)*m:((2*i)+1)*m],2),int(bitstr[((2*i
                )+1)*m:((2*i)+2)*m],2)))
141     return a
142
143 def seleccionaConjuntoAleatoriodeAcceso(partes,k):
144     """
```

```
145     Esta funcion selecciona un conjunto de acceso aleatorio de las
146     partes generadas.
147
148     @param partes: partes generadas del secreto.
149     @param k: umbral de acceso.
150
151     @return: vector cuyos elementos son las partes que conforman el
152         conjunto de acceso.
153     """
154     rnd.shuffle(partes)
155     return partes[:rnd.randint(k,len(partes))]
156
157 def seleccionaConjuntodeAcceso(partes,t):
158     """
159     Esta funcion selecciona un conjunto de acceso con <t> elementos
160     de las partes generadas.
161
162     @param partes: partes generadas del secreto
163     @param t: umbral de acceso
164
165     @return: vector cuyos elementos son las partes que conforman el
166         conjunto de acceso
167     """
168     rnd.shuffle(partes)
169     return partes[:rnd.randint(1,t)]
170
171
172 def evaluaInterpolaciondeLagrange(campo, partes, pol):
173     """
174     Esta funcion obtiene el valor constante de un polinomio con coeficientes
175     en un campo <campo> construido con interpolacion de Lagrange con puntos
176     dados (<partes>).
177
178     @param campo: campo sobre el que se encuentran los coeficientes del polinomio.
179     @param partes: conjunto de partes con los puntos que se usaran para
180         reconstruir el polinomio.
181     @param pol: indicador para saber que puntos de las partes se usaran
182         para construir el polinomio. (0 para los secretos, 2 para los
183         valores verificadores)
184
185     @return suma: termino constante del polinomio construido.
186     """
187     terminos= []
188     for i in range(0,len(partes)):
189         num = 1
190         den = 1
191         for j in range(0,len(partes)):
192             if j != i:
193                 den = campo.mult(den,campo.add(partes[i][pol],partes[j][pol]))
194                 num = campo.mult(num,campo.add(0,partes[j][pol]))
195         den = campo.inv(den)
196         terminos.append(campo.mult(num,campo.mult(partes[i][pol+1],den)))
197     suma = terminos[0]
198     for i in range(1,len(terminos)):
199         suma = campo.add(suma,terminos[i])
200     return suma
```

```
201
202 def shareGen(secreto, participantes, umbral, q1, campo1, q2, campo2, m, N, Prop):
203     """
204     Esta funcion genera las partes que se repartiran entre los
205     participantes del esquema.
206
207     @param secreto: secreto a compartir.
208     @param participantes: numero de participantes del esquema.
209     @param umbral: tamano minimo de los conjuntos de acceso.
210     @param q1: numero de elementos del campo <campo1>.
211     @param campo1: campo donde esta definido el secreto.
212     @param q2: numero de elementos del campo <campo2>.
213     @param campo2: campo donde esta definido el valor verificador.
214     @param m: tamano de los segmentos que se usaran para calcular el digito
215         verificador.
216     @param N: numero de segmentos en que se partira el secreto para
217         calcular el valor verificador.
218
219     @return a: valor del digito verificador construido.
220     @return ss: conjunto de partes que se repartira entre los participantes
221     """
222     #se construye el polinomio
223     F = polinomioAleatorio(umbral-1, secreto, q1)
224     #se define el valor verificador y se construye el polinomio verificador
225     a = construyeValorVerificador(campo2, secreto, m, N, Prop)
226     G = polinomioAleatorio(umbral-1, a, q2)
227     #se generan las partes a compartir
228     ss = []
229     ssi = []
230     for i in range(0, participantes):
231         ssi.append(i+1)
232         ssi.append(evaluaPolinomio(F,i+1,campo1))
233         if Prop == 1 or Prop == 2:
234             ssi.append(i+1)
235             ssi.append(evaluaPolinomio(G,i+1,campo1))
236         else:
237             ssi.append(i+1)
238             ssi.append(evaluaPolinomio(G,i+1,campo2))
239         ss.append(ssi)
240         ssi = []
241     return a, ss
242
243 def tpolinomio(ca,grado):
244     """
245     Esta funcion evalua si el conjunto de acceso es de un tamano valido.
246
247     @param ca: conjunto de acceso.
248     @param grado: grado del polinomio definido en ShareGen.
249
250     @return: resultado de la evaluacion.
251     """
252     if len(ca) <= grado:
253         return False
254
255     for i in range(0,len(ca)-2):
256         for j in range(i+1,len(ca)-1):
```

```
257             if ca[i] == ca[j]:
258                 return False
259     return True
260
261 def reconst(campo1, campo2, conjunto, m, N, Prop, k):
262     """
263     Esta funcion reconstruye el secreto y el digito verificador
264     con las partes del conjunto que intenta reconstruir el secreto.
265     Ademas verifica si hubo algun problema con una o mas de las partes.
266
267     @param campo1: campo sobre el cual esta definido el secreto.
268     @param campo2: campo sobre el que esta definido el digito verificador.
269     @param conjunto: conjunto de partes con las que se pretende
270         reconstruir el secreto.
271     @param m: tamano de los segmentos en que se partira el secreto para
272         calcular el digito verificador.
273     @param N: numero de segmentos en que se partira el secreto para
274         calcular el digito verificador.
275
276     @return: secreto reconstruido en caso de que la reconstruccion haya
277         sido exitosa, o mensaje de error (--) si alguien hizo trampa, o el
278         conjunto de partes no es suficiente para reconstruir el secreto.
279     """
280     s1 = evaluaInterpolaciondeLagrange(campo1, conjunto, 0)
281     if Prop==1:
282         if (not tpolinomio(conjunto,k-1)) :
283             return "--"
284     a1 = evaluaInterpolaciondeLagrange(campo2, conjunto, 2)
285     a2 = construyeValorVerificador(campo2, s1, m, N, Prop)
286     if a1 == a2:
287         return str(s1)
288     else:
289         return "--"
290
291 def pruebaCAV(campo1, campo2, s, p, k, q1, q2, N, m):
292     """
293     Esta funcion prueba el desempeno de los esquemas de Hoshino-Obana y
294     Becerra-Vega de acuerdo a un proceso ideal de recuperacion del secreto.
295
296     @param campo1: campo sobre el cual esta definido el secreto.
297     @param campo2: campo sobre el que esta definido el digito verificador.
298     @param s: secreto a compartir.
299     @param k: tamano del umbral.
300     @param q1: numero de elementos del campo <campo1>.
301     @param q2: numero de elementos del campo <campo2>.
302     @param N: numero de segmentos en que se partira el secreto para
303         calcular el digito verificador.
304     @param m: tamano de los segmentos en que se partira el secreto para
305         calcular el digito verificador.
306
307     @return rs1, rs2: resultado del desempeno de los esquemas, (0: Trampa
308         satisfactoria, 1: Ejecucion limpia, 2: Error [trampa o CANV]).
309     """
310     #Se generan las partes con el esquema propuesto (1) y con el esquema de Hoshino y
            Obana (0)
311     a1, Partes1 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,1)
```

```
312     a2, Partes2 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,0)
313     #Se generan conjuntos de acceso aleatorios validos
314     conjuntoA1 = seleccionaConjuntoAleatoriodeAcceso(Partes1,k)
315     conjuntoA2 = seleccionaConjuntoAleatoriodeAcceso(Partes2,k)
316     #Se intenta recuperar el secreto
317     resultadoA1 = reconst(campo1,campo2,conjuntoA1,m,N,1,k)
318     resultadoA2 = reconst(campo1,campo2,conjuntoA2,m,N,0,k)
319     rs1=0
320     rs2=0
321     if "--" == resultadoA1:
322         rs1 = 2
323     if "--" == resultadoA2:
324         rs2 = 2
325     if str(s) == resultadoA1:
326         rs1 = 1
327     if str(s) == resultadoA2:
328         rs2 = 1
329     return rs1, rs2
330
331 def pruebaCANV(campo1, campo2, s, p, k, q1, q2, N, m):
332     """
333     Esta funcion prueba el desempeno de los esquemas de Hoshino-Obana y
334     Becerra-Vega de acuerdo a un proceso de recuperacion del secreto en el que
335     el conjunto de acceso no es valido.
336
337     @param campo1: campo sobre el cual esta definido el secreto.
338     @param campo2: campo sobre el que esta definido el digito verificador.
339     @param s: secreto a compartir.
340     @param k: tamano del umbral.
341     @param q1: numero de elementos del campo <campo1>.
342     @param q2: numero de elementos del campo <campo2>.
343     @param N: numero de segmentos en que se partira el secreto para
344         calcular el digito verificador.
345     @param m: tamano de los segmentos en que se partira el secreto para
346         calcular el digito verificador.
347
348     @return rs1, rs2: resultado del desempeno de los esquemas, (0: Trampa
349         satisfactoria, 1: Ejecucion limpia, 2: Error [trampa o CANV]).
350     """
351     #Se generan las partes con el esquema propuesto (1) y con el esquema de Hoshino y
        Obana (0)
352     a1, Partes1 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,1)
353     a2, Partes2 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,0)
354     #Se generan conjuntos de acceso aleatorios no validos
355     conjuntoB1 = seleccionaConjuntodeAcceso(Partes1,k-1)
356     conjuntoB2 = seleccionaConjuntodeAcceso(Partes2,k-1)
357     #Se intenta recuperar el secreto
358     resultadoB1 = reconst(campo1,campo2,conjuntoB1,m,N,1,k)
359     resultadoB2 = reconst(campo1,campo2,conjuntoB2,m,N,0,k)
360     rs1=0
361     rs2=0
362     if "--" == resultadoB1:
363         rs1 = 2
364     if "--" == resultadoB2:
365         rs2 = 2
366     if str(s) == resultadoB1:
```

```
367             rs1 = 1
368         if str(s) == resultadoB2:
369             rs2 = 1
370         return rs1, rs2
371 
372 def pruebaT(campo1, campo2, s, p, k, q1, q2, N, m):
373         """
374         Esta funcion prueba el desempeno de los esquemas de Hoshino-Obana y
375         Becerra-Vega de acuerdo a un proceso de recuperacion del secreto en el que
376         el conjunto de acceso es valido pero continene una parte que fue modificada.
377 
378         @param campo1: campo sobre el cual esta definido el secreto.
379         @param campo2: campo sobre el que esta definido el digito verificador.
380         @param s: secreto a compartir.
381         @param k: tamano del umbral.
382         @param q1: numero de elementos del campo <campo1>.
383         @param q2: numero de elementos del campo <campo2>.
384         @param N: numero de segmentos en que se partira el secreto para
385             calcular el digito verificador.
386         @param m: tamano de los segmentos en que se partira el secreto para
387             calcular el digito verificador.
388 
389         @return rs1, rs2: resultado del desempeno de los esquemas, (0: Trampa
390             satisfactoria, 1: Ejecucion limpia, 2: Error [trampa o CANV]).
391         """
392         #Se generan las partes con el esquema propuesto (1) y con el esquema de Hoshino y
                Obana (0)
393         a1, Partes1 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,1)
394         a2, Partes2 = shareGen(s,p,k,q1,campo1,q2,campo2,m,N,0)
395         #Se generan conjuntos de acceso aleatorios validos
396         conjuntoC1 = seleccionaConjuntoAleatoriodeAcceso(Partes1,k)
397         conjuntoC2 = seleccionaConjuntoAleatoriodeAcceso(Partes2,k)
398         #algunos hacen trampa:
399         nt=rnd.randint(1,k-2)
400         for i in range(0,nt):
401             conjuntoC1[i][1] = elementoAleatorio(q1)
402             conjuntoC2[i][1] = elementoAleatorio(q1)
403         #Se intenta recuperar el secreto
404         resultadoC1 = reconst(campo1,campo2,conjuntoC1,m,N,1,k)
405         resultadoC2 = reconst(campo1,campo2,conjuntoC2,m,N,0,k)
406         rs1=0
407         rs2=0
408         if "--" == resultadoC1:
409             rs1 = 2
410         if "--" == resultadoC2:
411             rs2 = 2
412         if str(s) == resultadoC1:
413             rs1 = 1
414         if str(s) == resultadoC2:
415             rs2 = 1
416         return rs1, rs2
417 
418 def iteracion(q1,GF2mN,q2,GF2m,umbral,numpar,N,m,RCV1,RCV2,RCNV1,RCNV2,RT1,RT2):
419         """
420         Esta funcion realiza las tres pruebas disenadas para los parametros
421         indicados.
```

```
422
423     @param q1: numero de elementos del campo <GF2mN>.
424     @param campo1: campo sobre el cual esta definido el secreto.
425     @param q2: numero de elementos del campo <GF2m>.
426     @param campo2: campo sobre el que esta definido el digito verificador.
427     @param umbral: tamano del umbral.
428     @param numpar: numero de participantes del esquema.
429     @param N: numero de segmentos en que se partira el secreto para
430         calcular el digito verificador.
431     @param m: tamano de los segmentos en que se partira el secreto para
432         calcular el digito verificador.
433     @param RCV1: vector con los resultados de la primer prueba aplicada al
434         primer esquema en las iteraciones previas.
435     @param RCV2: vector con los resultados de la primer prueba aplicada al
436         segundo esquema en las iteraciones previas.
437     @param RCNV1: vector con los resultados de la segunda prueba aplicada al
438         primer esquema en las iteraciones previas.
439     @param RCNV2: vector con los resultados de la segunda prueba aplicada al
440         segundo esquema en las iteraciones previas.
441     @param RT1: vector con los resultados de la tercer prueba aplicada al
442         primer esquema en las iteraciones previas.
443     @param RT1: vector con los resultados de la tercer prueba aplicada al
444         segundo esquema en las iteraciones previas.
445
446     @return RCV1,RCV2,RCNV1,RCNV2,RT1,RT2: vectores con el conteo de los
447         resultados de las pruebas realizadas incluyendo los de esta ejecucion.
448     """
449     #se define el secreto
450     secreto = elementoAleatorio(q1)
451     a,b = pruebaCAV(GF2mN, GF2m, secreto, numpar, umbral, q1, q2, N, m)
452     RCV1[a]+=1
453     RCV2[b]+=1
454     a,b = pruebaCANV(GF2mN, GF2m, secreto, numpar, umbral, q1, q2, N, m)
455     RCNV1[a]+=1
456     RCNV2[b]+=1
457     a,b = pruebaT(GF2mN, GF2m, secreto, numpar, umbral, q1, q2, N, m)
458     RT1[a]+=1
459     RT2[b]+=1
460     return RCV1,RCV2,RCNV1,RCNV2,RT1,RT2
461
462 ##############################################################################
463 #se indica el numero de iteraciones a realizar
464 NumPruebas = 100
465
466 #se definen los parametros de los esquemas
467 vnumpar = 6    #Numero de Participantes
468 vumbral = 3    #Tamano del umbral
469 vm = 4    #Tamano de los segmentos del secreto
470 vN = 4    #Numero de segmentos del secreto
471
472 #el tamano de los campos debe ser mayor al numero de participantes
473 vq1 = 2**(vm*vN)
474 vq2 = 2**vm
475
476 #se construyen los campos sobre los que se trabajara
477 vGF2mN = GF2(vm*vN)
```

```
vGF2m = GF2(vm)

RCV1=np.zeros(3)
RCV2=np.zeros(3)
RCNV1=np.zeros(3)
RCNV2=np.zeros(3)
RT1=np.zeros(3)
RT2=np.zeros(3)

#Se realizan las pruebas
for i in range(0,NumPruebas):
    RCV1,RCV2,RCNV1,RCNV2,RT1,RT2 = iteracion(vq1,vGF2mN,vq2,vGF2m,vumbral,vnumpar,vN,
        vm,RCV1,RCV2,RCNV1,RCNV2,RT1,RT2)

#Se calculan los porcentajes
RCV1=100*RCV1/NumPruebas
RCV2=100*RCV2/NumPruebas
RCNV1=100*RCNV1/NumPruebas
RCNV2=100*RCNV2/NumPruebas
RT1=100*RT1/NumPruebas
RT2=100*RT2/NumPruebas

#Se imprimen los resultados
Nc = 6
prueba1 = (RCV1[1], RCV2[1], 0, 0, 0, 0)
prueba2 = (0, 0, RCNV1[2], RCNV2[2], 0, 0)
prueba3 = (0, 0, 0, 0, RT1[2], RT2[2])
Std = (0, 0, 0, 0, 0, 0)
ind = np.arange(Nc)
indl = np.arange(Nc/2)
width = 0.7
plt.figure(figsize=(8,5))
patterns = ('', '//', '', '//', '', '//','', '//', '', '//', '', '//','', '//', '', '
    //', '', '//')
bar1 = plt.bar(ind, prueba1, width, color='dimgray', edgecolor='darkgrey', yerr=Std,
    label='Secreto recuperado en uso ideal')
bar2 = plt.bar(ind, prueba2, width,
            bottom=prueba1, color='darkgray', edgecolor='white', yerr=Std, label='
                Deteccion del conjunto de acceso invalido')
bar3 = plt.bar(ind, prueba3, width,
            bottom=prueba2, color='lightgray', edgecolor='white', yerr=Std, label='
                Deteccion de trampa')
bars = bar1 + bar2 + bar3
plt.ylabel('Porcentaje')
plt.xticks([])
plt.yticks(np.arange(0, 101, 10))
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.08), ncol=1)
for bar, pattern in zip(bars, patterns):
    bar.set_hatch(pattern)
for ibar1, ibar2, ibar3 in zip(bar1, bar2, bar3):
    h1 = ibar1.get_height()
    h2 = ibar2.get_height()
    h3 = ibar3.get_height()
    if h1 != 0 :
        plt.text(ibar1.get_x() + ibar1.get_width() / 2., h1 / 2., "%g%% %h1, ha="
            center", va="center", color="black", fontsize=10.5)
```

```
528     if h2 != 0 :
529         plt.text(ibar2.get_x() + ibar2.get_width() / 2., h1 + h2 / 2., "%g%%" %h2, ha
                ="center", va="center", color="black", fontsize=10.5)
530     if h3 != 0 :
531         plt.text(ibar3.get_x() + ibar3.get_width() / 2., h1 + h2 + h3 / 2., "%g%%" %
                h3, ha="center", va="center", color="black", fontsize=10.5)
532 plt.show()
```

Apéndice B

Actividades derivadas del presente trabajo

A continuación se incluyen las constancias de las conferencias impartidas en las cuales se presentó el contenido del presente trabajo, particularmente, el esquema propuesto en el Capítulo 2, así como el certificado del premio otorgado por el comité organizador de "The 2nd International Conference on Networking, Information Systems & Security" al artículo *Secret Sharing Scheme With Efficient Cheating Detection*.

SEGURIDAD EN CÓMPUTO
UNA VISIÓN ACADÉMICA

El Instituto de Matemáticas Aplicadas y en Sistemas y
el Posgrado en Ciencia e Ingeniería de la Computación

Otorgan la presente

Constancia

a

Ing. Daniel Becerra Pedraza

Por su destacada participación en la impartición de la conferencia

"¿Cómo compartir un secreto con tramposos?"

Ciudad Universitaria, Cd. Mx.
7 de Noviembre de 2018

Dr. Héctor Benítez Pérez
Director del Instituto de Investigaciones en
Matemáticas Aplicadas y en Sistemas

Dr. Javier Gómez Castellanos
Coordinador del Posgrado en Ciencia e
Ingeniería de la Computación

UNAM

Figura B-1: Constancia de la conferencia *¿Cómo compartir un secreto con tramposos?*.

Figura B-2: Certificado del premio otorgado a [2].

Figura B-3: Constancia de la conferencia *Esquema de Compartición de Secretos con Detección Eficiente de Trampas*.

Bibliografía

[1] H. Hoshino and S. Obana, "Cheating detectable secret sharing scheme suitable for implementation," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pp. 623–628, Nov 2016.

[2] D. Becerra and G. Vega, "Secret sharing scheme with efficient cheating detection," in *Proceedings of the 2Nd International Conference on Networking, Information Systems & Security*, NISS19, (New York, NY, USA), pp. 5:1–5:7, ACM, 2019.

[3] G. J. Simmons, "How to (really) share a secret," in *Advances in Cryptology — CRYPTO' 88* (S. Goldwasser, ed.), (New York, NY), pp. 390–448, Springer New York, 1990.

[4] C. L. Liu, *Introduction to combinatorial mathematics*. McGraw-Hill, 1968.

[5] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.

[6] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on(AFIPS)*, vol. 00, p. 313, 12 1899.

[7] M. Tompa and H. Woll, "How to share a secret with cheaters," *Journal of Cryptology*, vol. 1, pp. 133–138, Oct 1989.

[8] B. Schneier, *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995.

[9] S. Obana and K. Tsuchida, "Cheating detectable secret sharing schemes supporting an arbitrary finite field," in *Advances in Information and Computer Security* (M. Yoshida and K. Mouri, eds.), (Cham), pp. 88–97, Springer International Publishing, 2014.

[10] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, (Washington, DC, USA), pp. 383–395, IEEE Computer Society, 1985.

[11] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Commun. ACM*, vol. 24, pp. 583–584, Sept. 1981.

[12] C. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE Transactions on Information Theory*, vol. 29, pp. 208–210, March 1983.

[13] Á. Rey, J. Mateus, and G. Sánchez, "A secret sharing scheme based on cellular automata," *Applied Mathematics and Computation*, vol. 170, pp. 1356–1364, 11 2005.

[14] A. Adhikari, K. Morozov, S. Obana, P. S. Roy, K. Sakurai, and R. Xu, "Efficient threshold secret sharing schemes secure against rushing cheaters," in *Information Theoretic Security* (A. C. Nascimento and P. Barreto, eds.), (Cham), pp. 3–23, Springer International Publishing, 2016.

[15] Z. Chen, S. Li, Y. Zhu, J. Yan, and X. Xu, "A cheater identifiable multi-secret sharing scheme based on the chinese remainder theorem," *Sec. and Commun. Netw.*, vol. 8, pp. 3592–3601, Dec. 2015.

[16] Y. Liu, Z. Wang, and W. Yan, "Linear (k, n) secret sharing scheme with cheating detection," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pp. 1942–1947, Oct 2015.

[17] Y. Liu, C. Yang, Y. Wang, L. Zhu, and W. Ji, "Cheating identifiable secret sharing scheme using symmetric bivariate polynomial," *Information Sciences*, vol. 453, pp. 21–29, 2018.

[18] N. Singh, A. N. Tentu, A. Basit, and V. C. Venkaiah, "Sequential secret sharing scheme based on chinese remainder theorem," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–6, Dec 2016.

[19] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Advances in Cryptology CRYPTO' 99* (M. Wiener, ed.), (Berlin, Heidelberg), pp. 148–164, Springer Berlin Heidelberg, 1999.

[20] Q. Zhao and Y. Liu, "E-voting scheme using secret sharing and k-anonymity," in *Advances on Broad-Band Wireless Computing, Communication and Applications* (L. Barolli, F. Xhafa, and K. Yim, eds.), Springer International Publishing, 2017.

[21] A. Krishnan and M. L. Das, "Medical image security with cheater identification using secret sharing scheme," in *Proceedings of the International Conference on Signal, Networks, Computing, and Systems* (D. K. Lobiyal, D. P. Mohapatra, A. Nagar, and M. N. Sahoo, eds.), (New Delhi), pp. 117–126, Springer India, 2017.

[22] Y.-X. Liu, Q.-D. Sun, and C.-N. Yang, "(k,n) secret image sharing scheme capable of cheating detection," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, p. 72, Apr 2018.

[23] L. Harn and C.-F. Hsu, "A Practical Hybrid Group Key Establishment for Secure Group Communications," *The Computer Journal*, vol. 60, pp. 1582–1589, 01 2017.

[24] M. A. Hadavi, R. Jalili, E. Damiani, and S. Cimato, "Security and searchability in secret sharing-based data outsourcing," *International Journal of Information Security*, vol. 14, pp. 513–529, Nov 2015.

[25] M. A. Hadavi, R. Jalili, and L. Karimi, "Access control aware data retrieval for secret sharing based database outsourcing," *Distributed and Parallel Databases*, vol. 34, pp. 505–534, Dec 2016.

[26] J. Wang, C. Huang, N. N. Xiong, and J. Wang, "Blocked linear secret sharing scheme for scalable attribute based encryption in manageable cloud storage system," *Information Sciences*, vol. 424, pp. 1 – 26, 2018.

[27] W.-i. Bae and J. Kwak, "Smart card-based secure authentication protocol in multi-server iot environment," *Multimedia Tools and Applications*, Dec 2017.

[28] N. Park and D. Lee, "Electronic identity information hiding methods using a secret sharing scheme in multimedia-centric internet of things environment," *Personal and Ubiquitous Computing*, vol. 22, pp. 3–10, Feb 2018.

[29] K.-H. Wang, C.-M. Chen, W. Fang, and T.-Y. Wu, "On the security of a new ultra-lightweight authentication protocol in iot environment for rfid tags," *The Journal of Supercomputing*, vol. 74, pp. 65–70, Jan 2018.

[30] S. Banerjee, D. S. Gupta, and G. P. Biswas, "Hierarchy-based cheating detection and cheater identification in secret sharing schemes," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, pp. 1–6, March 2018.

[31] S. Cabello, C. Padró, and G. Sáez, "Secret sharing schemes with detection of cheaters for a general access structure," *Des. Codes Cryptography*, vol. 25, pp. 175–188, Feb. 2002.

[32] M. Carpentieri, A. De Santis, and U. Vaccaro, "Size of shares and probability of cheating in threshold schemes," in *Advances in Cryptology — EUROCRYPT '93* (T. Helleseth, ed.), (Berlin, Heidelberg), pp. 118–125, Springer Berlin Heidelberg, 1994.

[33] W. Ogata, K. Kurosawa, and D. Stinson, "Optimum secret sharing scheme secure against cheating," *SIAM Journal on Discrete Mathematics*, vol. 20, no. 1, pp. 79–95, 2006.

[34] T. Araki and W. Ogata, "A simple and efficient secret sharing scheme secure against cheating," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E94.A, no. 6, pp. 1338–1345, 2011.

[35] H. Hoshino and S. Obana, "Almost optimum secret sharing schemes with cheating detection for random bit strings," in *Advances in Information and Computer Security* (K. Tanaka and Y. Suga, eds.), (Cham), pp. 213–222, Springer International Publishing, 2015.

[36] K. Ding and C. Ding, "A class of two-weight and three-weight codes and their applications in secret sharing," *CoRR*, vol. abs/1503.06512, 2015.

[37] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. New York, NY, USA: Cambridge University Press, 1986.

[38] R. Lidl and H. Niederreiter, *Finite Fields*. New York, NY, USA: Cambridge University Press, 1997.

[39] A. for Applications, *Slinko, Arkadii*. Springer International Publishing, 2015.

[40] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1974.

[41] R. M. Capocelli, A. De Santis, L. Gargano, and U. Vaccaro, "On the size of shares for secret sharing schemes," *Journal of Cryptology*, vol. 6, pp. 157–167, Mar 1993.

[42] C. Blundo, A. D. Santis, R. D. Simone, and U. Vaccaro, "Tight bounds on the information rate of secret sharing schemes," *Designs, Codes and Cryptography*, vol. 11, pp. 107–110, May 1997.