ITBA - "72.44 - Criptografía y Seguridad"

Trabajo Práctico de Implementación: 'Secreto Compartido con Esteganografía'

Alumnos:

De Simone, Franco	61100
Dizenhaus, Manuel	61101
Negro, Juan Manuel	61225
Sambartolomeo, Mauro Daniel	61279

Índice

1. Introducción

Este trabajo parte de los fundamentos de lo que se conoce como criptografía visual, concepto introducido por Adi Shamir y Moni Naor en 1994. El concepto se basa en la idea de que información visual puede ser encriptada de tal forma que la desencripción resulte también en una información visual.
Basándose en el Esquema de Secreto Compartido (k, n)[1], también propuesto por Shamir, sumado a la utilización de métodos de esteganografía, se presenta el *paper* a implementar: *"(k,n) secret image sharing scheme capable of cheating detection"*[2], elaborado por Yan-Xiao Liu, Quin-Dong Sun y Ching-Nung Yang de la Universidad de Tecnología de Xi'an (China).
En el mismo se detalla cómo se puede ocultar una imagen utilizando n sombras, de forma tal que se requieren como mínimo k sombras para obtener la imagen original. El esquema propuesto extiende el esquema de Shamir, aportando capacidad de detectar la presencia de sombras falsas.
Siguiendo los lineamientos de este *paper*, se implementó un sistema que es capaz de ocultar una imagen de formato *BMP*, obteniendo las n sombras requeridas, y luego ocultarlas en otras imágenes del mismo formato. Luego, se implementó la función inversa, donde se reconstruye la imagen original con las k sombras necesarias, detectando sombras falsas.

[1]Shamir, Adi (1979), "*How to share a secret*", *Communications of the ACM*, Volumen 22 (Número 11): p612–613. Disponible en https://dl.acm.org/doi/10.1145/359168.359176 (Visitado Junio 2023)

[2]Liu, X., Sun, Q., Yang C. "*(k,n) secret image sharing scheme capable of cheating detection*", *EURASIP Journal on Wireless Comunication and Networking* ,2018. Disponible en https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-018-1084-7 (Visitado Junio 2023)

2. Modelo matemático

Como se mencionó en la introducción, el *paper* utiliza el Esquema de Secreto Compartido (k, n) desarrollado por Adi Shamir, que permite distribuir un secreto en n sombras, necesitando al menos k para recuperar el mismo. El mismo utiliza el método interpolador de Lagrange para reconstruir polinomios en base a las sombras otorgadas.
Es importante mencionar que este *paper* trabaja con operatoria modular, con el módulo siendo 251.

2.1. Distribución de una imagen secreta

El *paper* parte de una imagen secreta I, y realiza los siguientes pasos:

1. Se divide I en t bloques de $2k-2$ píxeles.
2. Para cada bloque t, tenemos $2k-2$ píxeles secretos $a_{i,0}, ..., a_{i,k-1}$ y $b_{i,2}, ..., b_{i,k-1}$. Con los primeros k píxeles (con su valor en *bytes*), generan el polinomio $f_i(x) = a_{i,0} + a_{i,1}x + , \ldots , + a_{i,k-1}x^{k-1} \in GF(251)[X]$.
3. Luego, el *dealer* selecciona un entero r_i al azar, y computa dos píxeles $b_{i,0}, b_{i,1}$ tales que cumplan que:

   $$r_i a_{i,0} + b_{i,0} = 0, r_i a_{i,1} + b_{i,1} = 0$$

   y con ellos genera otro polinomio de grado $k-1$ de forma $g_i(x) = b_{i,0} + b_{i,1}x + , \ldots , + b_{i,k1}x^{k1} \in GF(251)[X]$
4. Para cada bloque $B_i, i \in [1, t]$, se genera una subsombra $v_{i,j} = \{m_{i,j}, d_{i,j}\}$, con $m_{i,j} = f_i(j)$ y $d_{i,j} = g_i(j)$, $j \in [1, n]$.
5. Finalmente, la sombra S_j se constituye por $S_j = v_{1,j}||...||v_{t,j}$

Nos quedan las N sombras, donde necesitamos al menos k para recuperar el secreto.

2.2. Reconstrucción de la imagen

Para recuperar una imagen secreta I, es necesario recordar que al momento de distribuirla, se tomo un valor de k que determinó la cantidad mínima de sombras para hallar la original.
Partiendo de esto, tenemos k sombras $S_1, ..., S_k$

1. Extraigo los $v_{i,j} = (m_{i,j}, d_{i,j})$ con $i \in [1, t], j \in [1, k]$ de cada sombra S_j
2. Para cada $v_{i,1}, ..., v_{i,k}$, reconstruimos $f_i(x), g_i(x)$ a partir de los $m_{i,1}, ..., m_{i,k}$ y $d_{i,1}, ..., d_{i,k}$ respectivamente, utilizando el método de Interpolación de Lagrange[3].
3. Sean $a_{i,0}, a_{i,1}, b_{1,0}, b_{i,1}$ los coeficientes de x^0 y x en $f_i(x), g_i(x)$.
   - Si existe un r_i que verifique $r_i a_{i,0} + b_{i,0} = 0$ y $r_i a_{i,1} + b_{i,1} = 0$, entonces recuperamos el bloque $B_i = \{a_{i,0}, ..., a_{i,k-1}, b_{i,2}, b_{1,3}, ..., b_{i,k-1}\}$. Realizamos lo mismo para todos los bloques.
   - Si este r_i no existe, entonces estamos lidiando con sombras falsas, se detectó un caso de *cheating*, y se cancela la recuperación.
4. En caso que no se haya detectado *cheating*, la imagen recuperada I es $I = B_1||...||B_t$

[3]El método de Lagrange es utilizado en los fundamentos de Shamir. Para leer mas sobre esto, dirigirse al siguiente link

3. Implementación

La implementación fue llevada a cabo en lenguaje C, específicamente en su versión C23. El formateo del código se realizó con clang, una herramienta de código *open source*.[4].
El trabajo requiere de dos dependencias, utiliza `gcc` o `clang` para compilar el código y también `make` para facilitar la compilación para el usuario. Para compilarlo, alcanza simplemente con colocarse sobre la raíz del proyecto y correr `make all`. Esto generará la carpeta `target` en el directorio raíz del proyecto, donde se encontrarán tanto los archivos objeto como el ejecutable `shared_secret_steganography`.
Para ejecutar el proyecto, tenemos las dos funcionalidades propuestas en el *paper*:

- Distribuir una imagen secreta en n sombras, ocultas en imágenes portadoras.
- Reconstruir la imagen secreta a partir de k sombras.

3.1. Distribución

Para la distribución de la imagen secreta, se ejecuta el siguiente comando:

```
./target/shared_secret_steganography d <secreto.bmp> k
   <directorio donde se encuentran las imagenes a utilizar como
   sombras>
```

El programa leerá la imagen a esconder y generará n sombras, donde n es la cantidad de imágenes que se encuentren en el directorio especificado. Cada una de estas sombras tendrá un tamaño de $(k-1)^{-1}$ veces el tamaño de la imagen secreta, y se guardará ocultada en la imagen portadora correspondiente.
Este método de ocultamiento se realiza mediante el método de esteganografía *LSB* (*Least Significant Bit*), donde se modifican los X bits menos significativos de cada *byte*. En particular, se utiliza *LSB4* para valores de k iguales a 3 o 4, y *LSB2* para valores entre 5 y 8. De esta forma garantizamos que siempre se pueda ocultar cada sombra en la imagen portadora correspondiente.
Por último, se añade un número de sombra en el *header* de cada imagen portadora, para poder identificar qué sombra se encuentra en cada imagen. Esto se hace en el *offset* `0x06` del archivo, un espacio reservado de 2 *bytes* para la aplicación creadora de la imagen, por lo que lo utilizamos para guardar el número de sombra.

3.2. Reconstrucción

Para la reconstrucción del secreto, el comando a ejecutar es el siguiente:

```
./target/shared_secret_steganography r
   <path-donde-reconstruir.bmp> k <directorio donde se
   encuentran las imagenes a utilizar como sombras para la
   reconstrucci n>
```

El programa leerá el directorio de imágenes portadoras y tomará los primeros k archivos que encuentre, ignorando directorios y archivos que no sean `.bmp`. Luego, generará los polinomios descriptos en el *paper* y evaluará el caso de *cheating*, donde fallará si se detecta alguna sombra inválida. Luego, se reconstruirá la imagen secreta a partir de las sombras y se guardará en el nombre de imagen especificado, creando o reemplazando el archivo.

[4]Para mas información sobre compilación y requerimientos del proyecto, referirse al README del repositorio

3.2.1. Ejemplo - Reconstrucción de imágenes provistas por la cátedra

Fueron provistas al grupo 8 imágenes en las cuales se escondió un secreto con un esquema (4,8). Utilizamos nuestra implementación para obtener el secreto, el cual es el siguiente:

Figura 1: Imagen recuperada a partir de las sombras otorgadas

4. Cuestiones a analizar

1. Discutir los siguientes aspectos relativos al documento y al *paper*.

   - Organización formal del documento (¿es adecuada? ¿es confusa?)
   - La descripción del algoritmo de distribución y la del algoritmo de recuperación. (¿es clara? ¿es confusa? ¿es detallada? ¿es completa?)
   - La notación utilizada, ¿es clara? ¿cambia a lo largo del documento? ¿hay algún error?

   La organización del documento está orientada por secciones claramente delimitadas. Primero, comenta sobre los conocimientos preliminares necesarios para entender el algoritmo planteado. Estos son el esquema de secreto compartido desarrollado por Adi Shamir, el problema de detección de *cheating*[5] desarrollado por Martin Tompa y Heather Woll, y también el modelo para esconder imágenes[6]. Luego, procede a explicar el desarrollo del algoritmo, junto a las demostraciones matemáticas pertinentes. Finalmente, hay una sección de resultados, discusión, para luego cerrar con las conclusiones. El documento se encuentra estructurado como se espera de un trabajo académico, lo que facilita su seguimiento y consulta.

   En cuanto a la descripción del algoritmo y la notación, allí el documento por momentos utiliza notación similar para índices y subíndices sobre diferentes elementos, lo que genera confusión en el lector. Esto se debe al uso del índice de bloque de píxeles y el índice del píxel dentro de su mismo bloque, que si bien se utilizan correctamente, requiere una lectura detallada para su comprensión.

   Lo que llama la atención sobre el *paper* es que no hace alusión alguna al problema que tiene al utilizar módulo 251. Como se está trabajando con píxeles de 8 *bits*, los valores pueden ir de 0 a 255, generando pérdida de información para aquellos valores entre 251 y 255 dado que estamos trabajando con la operatoria modular.

2. El título del documento hace referencia a que es capaz de detectar sombras falsas (*cheating detection*). ¿cómo lo hace? ¿es un método eficaz?

   El *paper* propone un mecanismo para detectar sombras falsas al momento de recuperar cada bloque original. Esto lo realiza con el valor del entero aleatorio r_i seleccionado a la hora de generar las sombras.

   Este entero aleatorio debe funcionar tal que cumpla la condición indicada de $r_i a_{i,0} + b_{i,0} = 0$ y $r_i a_{i,1} + b_{i,1} = 0$ (se debe garantizar que ninguno de estos números sean 0). Luego, al reconstruir la imagen original se debe interpolar utilizando Lagrange, y así conseguir tanto $a_{i,j}$ como $b_{i,j}$, para $j = 0, 1$, y con ellos se puede hacer las siguientes operaciones modulares y conseguir el valor de r_i:

   $$r_{i,0} = (-b_{i,0}) * a_{i,0}^{-1}$$
   $$r_{i,1} = (-b_{i,1}) * a_{i,1}^{-1}$$
   $$¿r_{i,0} == r_{i,1}?$$

   Como se puede observar, se calculan dos candidatos al valor r_i, y solamente en el caso donde ambos sean iguales es cuando se puede afirmar que no se detectó *cheating*.

   Si se quiere pensar un ataque a este sistema, se podría pensar en un atacante en posesión de una sombra y logrando generar una sombra falsa que pase la detección analizada previamente. Para ello, el atacante debería modificar su imagen de forma que los $a_{i,0}$, $a_{i,1}$, $b_{i,0}$, y $b_{i,1}$ calculados mantengan las relaciones explicadas anteriormente. Como el atacante no tiene acceso directo a los $a_{i,0}$, $a_{i,1}$, $b_{i,0}$, y $b_{i,1}$ y solamente puede modificar las sombras de su imagen, esto seria muy difícil de conseguir sin acceso a las otras sombras.

[5] Tompa, M., Woll, H. (1989), "*How to share a secret with cheaters*"(https://link.springer.com/article/10.1007/BF02252871) (Visitado en Junio 2023)

[6] Thien, Chih-C., Lin, Ja-C., (2002), "*Secret image sharing*"(https://www.sciencedirect.com/science/article/abs/pii/S009784930200131 (Visitado en Junio 2023)

3. ¿Qué desventajas ofrece trabajar con congruencias módulo 251?

   La desventaja principal de trabajar con módulo 251 es que estamos trabajando con imágenes en blanco y negro. Esto implica que los valores de los píxeles se encuentran entre 0 y 255. Al trabajar con módulo 251, los *bytes* que tengan un valor de 251 o mayor, al distribuirse y recuperarse, van a tener su valor alterado. Es decir, si uno tiene un *byte* que vale 252, ocurre que $252 \equiv 1 \pmod{251}$, y el valor del píxel se "invierte", dado que pasa de un color prácticamente blanco a uno casi completamente negro.

   Esto se puede apreciar en las figuras 2 y 3. La figura 1 muestra la imagen original de Albert Einstein, una de las imágenes que se nos dio para probar el trabajo.

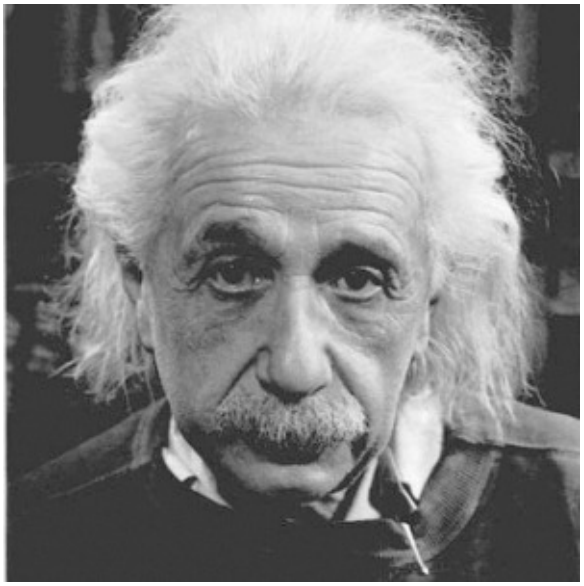

Figura 2: Imagen original

Luego, se tomo esta imagen *bmp* y se distribuyó en 9 sombras con un esquema (k,n), para luego recuperarla. El resultado fue el que se observa en la figura 3.

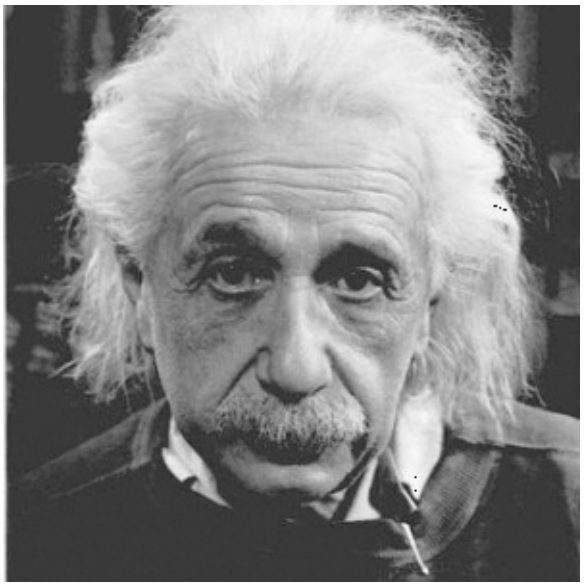

Figura 3: Imagen recuperada a partir de un esquema (3,9)

Como se puede observar en la figura 3, la imagen se recupera de manera correcta, con una particularidad: Sectores con píxeles que en la imagen original eran muy cercanos al blanco (es decir, tenían un valor entre 251 y 255) pasaron a ser negro, o valores muy chicos (entre 0 y 4). Esta diferencia se evidencia claramente en el cuello de la camisa de Albert, como también a la derecha de la imagen en su pelo.

Esto lógicamente se debe a lo explicado anteriormente, se produce por el efecto de trabajar con módulo 251.

4. Con este método, ¿se podrían guardar secretos de cualquier tipo (imágenes, pdf, ejecutables). ¿por qué? (relacionarlo con la pregunta 3)

   En el punto 3, observamos que existen limitaciones al trabajar con módulo 251. Al tener *bytes* que pueden ir hasta 255, hay información que se pierde. Sin embargo, al tratarse de $\frac{5}{256}$ posibles valores, la gran mayoría de la imagen se recupera de manera satisfactoria.

   Pero con otros tipos de información, el problema se complejiza aún mas. Sin ir mas lejos, consideremos imágenes en color. La información no se almacena en un *byte*, si no que se toman 3 *bits* para definir el color en su valor hexadecimal (Recordar que los colores siguen un formato de #XXXXXX, donde rojo, verde, y azul tienen 8 *bits* respectivamente). Trabajando exactamente con este método, considerando que cada píxel depende de estos 3 valores, se puede dar que los colores cambien rotundamente. En si, los valores que pueden cambiar son los mismos, pero el impacto de cambio en colores tiene un impacto mas grande, y puede alterar la coherencia de la imagen recuperada finalmente, dado que ahora son 3 *bytes* por píxel los que pueden perder información.

   En el caso de archivos binarios o documentos, las consecuencias son más graves. Al alterar *bytes*, se puede perder información que invalide completamente el uso de los mismos. Se debería buscar un mecanismo que asegure la no pérdida de información.

5. ¿En qué otro lugar o de qué otra manera podría guardarse el número de sombra?

   En primera instancia, se podría utilizar el otro espacio reservado en el *header*. Nosotros utilizamos `bfReserved1`, que tiene un offset en `0x06`, pero también podríamos haber optado por usar `bfReserved2`, cuyo offset es `0x08`.

   Otra forma de guardarlo seria mantener el número de sombra por fuera de la imagen y que sea otro input a la entrada del programa. En este caso se debería insertar el nombre del archivo que contiene las sombras junto a su número de sombra y estos se guardarían por separado. Este caso es más complejo y no tiene muchas ventajas en comparación con el que estamos utilizando.

6. ¿Qué ocurriría si se permite que r, a_0, a_1, b_0 o b_1 sean 0?

   Al encriptar las imágenes, se genera un `r` aleatorio entre 1 y 250 que cumpla que $r_i a_{i,0} + b_{i,0} = 0$ y $r_i a_{i,1} + b_{i,1} = 0$. Al permitir que r_i sea igual a 0, por como están planteadas las ecuaciones, $b_{i,0} = b_{i,1} = 0$. Esto no solo reduce la aleatoriedad de estos valores, si no que también le da la posibilidad al atacante de tener una mayor chance de introducir una sombra falsa, dado que tiene información sobre los coeficientes indicados.

   Lo mismo ocurre si $a_{i,0} = 0 \vee a_{i,1} = 0$. En cualquiera de los casos, condicionaría al $b_{i,p},\ p \in \{0, 1\}$, forzándolo a ser 0. Otra vez, eso le puede dar información potencial al atacante.

   Si $b_{i,0} = 0$ o $b_{i,1} = 0$, estamos ante un problema similar, dado que implica que $r_i a_{i,p} = 0$, lo que nos indica que tanto $r_i = 0$ o $a_{i,p} = 0$.

   Todos estos casos resultan en posibles puntos débiles para que un atacante explote. Recordemos que el reconocimiento de *cheating* es justamente validar que exista un r_i que cumpla las dos condiciones indicadas. Si r_i pudiera ser 0, alcanza con pasar sombras que tengan siempre $b_{i,0} = b_{i,1} = 0$, y el sistema no podrá detectar que se trata de una sombra falsa.

7. Explicar la relación entre el método de esteganografía (LSB2 o LSB4), el valor k y el tamaño del secreto y las portadoras.

   Siendo k el número de sombras creadas, se divide la imagen secreta I en t bloques de $2k-2$ bytes. En nuestro caso particular, la imagen original y las destino tienen el mismo tamaño de d *bytes* de alto por s *bytes* de largo.

   Considerando esto, al partir I, nos queda que $t = \frac{ds}{2k-2}$ (recordar que el tamaño de la imagen portadora es igual a la de la imagen secreta). De acuerdo a esta relación matemática, al aumentar k, se reduce la cantidad de bloques que se generan. Pero la cantidad de sombras generadas n permanece igual. Esto impacta a la hora de generar las subsombras $v_{i,j}$, dado que $j = 1, ..., n$.

   Por ende, se generará por cada bloque i, $2n$ *bytes* (correspondientes a la evaluación de los polinomios f y g para cada subsombra). Entonces, si tenemos una mayor cantidad de bloques (es decir, un k mas chico), existe una mayor cantidad de *bytes* para esconder. La consigna asegura que para $k \in \{3, 4\}$, el método esteganográfico $LSB4$ alcanzará para esconder todos los bytes generados. Y que, a partir de $k = 5$ en adelante, es necesario esconder utilizando el método $LSB2$.

8. Analizar cómo resultaría el algoritmo si se usaran imágenes en color (24bits por píxel) como portadoras.

   Siguiendo el concepto de la pregunta 4, las imágenes en color definen el color del mismo por su formato en hexadecimal, que ocupa 24 bits (3 bytes). Entonces, tenemos un formato `XXXXXX`, donde se definen los porcentajes de rojo, verde, y azul aplicados respectivamente.

   Como se explicó en la pregunta anterior, al querer esconder la información de estos bytes, ocurrirá lo mismo que con este algoritmo, y los bytes que tengan un valor entre 251 y 255, pasaran a estar entre 0 y 4, perdiendo así la información de los mismos. El problema se centra en que, al tratarse de 3 bytes diferentes para definir el color particular de un único píxel, se pierde una cantidad mayor de información que en el caso de la escala de grises.

9. Discutir los siguientes aspectos relativos al algoritmo implementado:

   - Facilidad de implementación
   - Posibilidad de extender el algoritmo o modificarlo.

   El algoritmo en si tiene la complejidad central de tener que trabajar con operatoria modular. Esto requiere un entendimiento matemático adecuado para poder analizar lo que está sucediendo a lo largo del desarrollo, como también para poder realizar un correcto *debuggeo*. Implementar la interpolación de *Lagrange* resultó un desafío interesante, con una complejidad de implementación superior al gran grueso del trabajo.

   Además, se debió investigar como se estructuran las imágenes de tipo *bmp*, para poder acomodar su encabezado a los usos que necesitábamos darle.

   En cuanto a una posible extensión del algoritmo, en nuestro caso de manejo de imágenes, se cuenta con una clara limitación dada por el formato de las imágenes a procesar (la manera en la que guardan su información, la escala de colores para cada píxel). En el caso de las imágenes en blanco y negro, el hecho de tener que trabajar en un espacio modular de 251 significa una restricción a posibles cambios en el algoritmo. Si se quisiera lidiar con imágenes de un formato diferente, tendría que estudiarse cuidadosamente el nuevo formato y modificar los algoritmos de distribución y recupero del secreto (particularmente el ocultamiento por esteganografía), cambiando las cuentas matemáticas involucradas para reflejar el nuevo espacio de trabajo.

10. ¿En qué situaciones aplicarían este tipo de algoritmos?

    Interpretando este tipo de algoritmos como aquellos que usan un esquema de secreto compartido para ocultar información, sus aplicaciones son infinitas. Ofrece la posibilidad de no ser

vulnerados ante la filtración de una sombra, por lo que para información sensible que comparten varias personas (por ejemplo, en una empresa) resulta interesante explorar potenciales aplicaciones.

Siguiendo la idea de la empresa, se ofrece una opción para controlar el poder de cada miembro de una comisión directiva, por ejemplo. Se puede necesitar una mayoría para realizar cierta acción, y a menos que esa parte decida accionar, entregando voluntariamente el secreto que le fue provisto, no se puede hacer uso de su voto.

5. Conclusiones

Se implementó exitosamente el algoritmo de *distribución de una imagen secreta mediante esquema (k, n) de secreto compartido con "cheating detection"* de acuerdo al paper provisto. Logramos distribuir una imagen en imágenes portadoras y usar algunas de ellas para recuperarla, al igual que recuperar un secreto desconocido previamente distribuido en imágenes portadoras por alguna entidad externa.

Este trabajo nos presentó la oportunidad de investigar mucho más acerca de aplicaciones de la criptografía, en especial de esquemas de secreto compartido, en el mundo real. Poder distribuir un secreto en distintas imágenes, sin que estas presenten, a simple vista, el hecho de haber sido modificadas y almacenar información secreta, se muestra como una herramienta muy poderosa para esconder información. Las operaciones matemáticas detrás del esquema han exigido un cuidado interesante a la hora de la implementación, con una complejidad significativa pero no abrumadora que resultó en un proceso rico y desafiante de desarrollo.

Finalmente pudimos verificar que, además de poder distribuir y recuperar un secreto con el esquema propuesto, se logra advertir la presencia de participantes maliciosos en el recupero de la información mediante el algoritmo de *cheating detection*. Para que el secreto pueda ser recuperado, se necesitan un mínimo de k imágenes portadoras, que deben ser exactamente aquellas en las que se distribuyó el secreto en primera instancia.