Análisis del esquema de secreto compartido capaz de detectar trampas

Autor

Ignacio Negro Caino - 57507 - inegro@itba.edu.ar

Introducción

En el presente documento se llevarán a cabo diversos análisis sobre el artículo científico titulado "Secret Image Sharing Scheme Capable of Cheating Detection", escrito por Yan-Xiao Liu, Qin-Dong Sun y Ching-Nung Yang. Además, se realizarán evaluaciones sobre el trabajo práctico de la asignatura Criptografía y Seguridad, específicamente respecto a la implementación de las ideas presentadas en dicho artículo mediante técnicas de esteganografía y los resultados obtenidos de ello.

Para finalizar, se exploraran ideas de extension, modificaciones y nuevas aplicaciones de esquema de secreto compartido.

1. Análisis del Paper

A continuación, se realizara un análisis sobre el paper “secret image sharing scheme capable of cheating detection” de Yan-Xiao Liu, Qin-Dong Sun y Ching-Nung Yang.

Con respecto a la organización del paper, los autores comienzan con un **abtract** en donde se presenta de manera resumida el problema el esquema, el problema de cheating, y la capacidad de esta extension de esquema de imagen secreta compartida de detectar comportamientos tramposos. Esta sección es clara, y da una buena idea de los que vamos a estar leyendo a continuación.

La siguiente sección es la **introducción**, se describe los esquemas de secretos compartidos sobre los cuales estará basada esta nueva version, importante y claro para tener contexto.

Luego tenemos la sección de “**Preliminaries**”, lo que vendría a ser otra introducción pero enfocada en comprender y explicar los algoritmos de esquema de Shamir, esquema de imagen secreta compartida de Thien-Lin, el cual vendría a ser el algoritmo base de este nuevo esquema.

Ademas, en esta sección se menciona por primera vez el escenario de “Tompa and Woll”, en donde multiples adversarios logran engañar a un participante honesto del esquema, logrando que el mismo recupere una imagen alterada, a través de haber provisto sombras falsas.

En la sección numero tres “(***k, n***) **secret image sharing with cheating detection**” se explica finalmente el nuevo esquema, en donde por primera vez, si se lo intenta seguir con detenimiento para realizar una implementación del mismo, se pueden notar algunos detalles que no son lo suficiente claros, lo que llevan al lector a asumir algunas cuestiones.

La primer inconsistencia a destacar aparece en el punto (3) de la fase de generación, en donde se explica como obtener los pixels $b_{i,0}$ y $b_{i,1}$, no se hace mención a que sucede si r_i, $a_{i,0}$, $a_{i,1}$, $b_{i,0}$ o $b_{i,1}$ fuesen cero, cuestión que se discutirá más adelante en este documento.

La segunda inconsistencia se presenta cuando se mencionan las sub-sombras, en la fase de generación, se las denota en el paso (4) como $v_{i,j} = \{m_{i,j}, d_{i,j}\}$, al final de este paso, se indica que una vez obtenida las sub-sombras para el participante P_j, se puede generar finalmente la sombra S_j como $S_j = v_{1,j}||...||v_{t,j}$. El problema con esta definición es que, no se hace referencia a como son codificados los valores de $m_{i,j}$ y $d_{i,j}$ para luego poder ser concatenados y formar la sombra correspondiente. Se podría interpretar incluso que los caracteres `{`, `,` y `}` forman parte de la sub-sombra.

En la sección de reconstrucción de la imagen, vuelve a surgir otra problema de nomenclatura con las sub-sombras, en el paso (1) ahora se las menciona como $v_{i,j} = (m_{i,j}, d_{i,j})$, es decir, con paréntesis en lugar de llaves. No se da ninguna explicación de ello lo cual se puede interpretar como un error. Nuevamente, al no haber mencionado anteriormente como fueron codificadas $m_{i,j}$ y $d_{i,j}$ para ser concatenadas en la sombra, se debe asumir como será la recuperación de las mismas.

Por ultimo, en el paso (3) del algoritmo, los autores vuelven a cometer otra error de escritura, se menciona el el bloque a recuperar es $B_i = \{a_{i,0}, ...a_{i,k-1}, b_{i,2}, ..., b_{i,k-1}\}$ lo cual en realidad debería haber sido $B_i = a_{i,0}||...||a_{i,k-1}||b_{i,2}||...||b_{i,k-1}$. En la explicación del esquema anterior “Thien-Lin’s secret image sharing” podemos notar que si se encuentra bien escrito.

En cuanto a la descripción del algoritmo, por lo mencionado anteriormente se puede concluir el mismo no es del todo claro y tampoco esta completo, ya que requiere asumir la codificación de las sub-sombras $v_{i,j}$, ademas de contar con ambigüedades en su escritura.

La sección numero cuatro, “**Results and discussion**” se ejemplifica la capacidad de detectar engaños del algoritmo, el ejemplo no es demostrado en detalle, de haber sido así, se podría haber detectado el problema anterior. De todas maneras, es util para tener una primera vision del esquema de manera practica.

Las siguientes secciones, “**Conclusions**” y “**Method**” se encuentran bien explicadas de forma clara y concisa, cumplen con su función de manera correcta.

2. Capacidad de detección de sombras falsas

En esta sección analizaremos la capacidad del esquema para detectar sombras falsas y su efectividad.

La capacidad de detectar engaños por parte de hasta $k-1$ adversarios se encuentra demostrada en el paper, específicamente en el **Teorema 2**.

En el mismo se explica que, si se tiene una lista de participantes $P_1, ..., P_k$ de los cuales $P_1, ..., P_{k-1}$ son adversarios, es decir, los mismos son poseedores de sombras falsas, el algoritmo seria capaz de detectar estas trampas al nivel de bloque, ya que, si recordamos el algoritmo de recuperación de imagen, en el paso (3) se realiza la prueba de falsedad sobre el bloque B_i a recuperar.

Ademas, debemos recordar que cada uno de las sombras esta formada por sub-sombras $v_{i,j}$ siendo j el numero de participante/sombra, por ende, teniendo $k-1$ adversarios, tendríamos $k-1$ sub-sombras falsas por bloque, por ende, como $v_{i,j} = (m_{i,j} = f_i(j), d_{i,j} = g_i(j))$, directamente las sub-sombras de los adversarios estarían generando puntos falsos sobre los polinomios, los cuales serán utilizados para, mediante interpolación, recuperar los coeficientes del polinomio original. Es decir, al interpolar puntos falsos, distintos a los originales, los polinomios resultantes también serian distintos a los originales.

Estos nuevos polinomios pueden pensarse como, el polinomio original sumado a la modificación introducida por el adversario, quedando como resultado $f^{**}(x) = f(x) + f^*(x)$ y $g^{**}(x) = g(x) + g^*(x)$, luego, $f^*(x) = a_0^* + a_1^* x + ... + a_{k-1}^* x^{k-1}$ y $g^*(x) = b_0^* + b_1^* x + ... + b_{k-1}^* x^{k-1}$ deberán haber sido construidos habiendo seleccionado un numero r^*, que cumpla las condiciones del algoritmo $r^* a_0^* + b_0^* = 0$ y $r^* a_1^* + b_1^* = 0$.

Dicho esto, para los adversarios poder lograr pasar la prueba de falsedad, deberán poder cumplir con $r(a_0 + a_0^*) + (b_0 + b_0^*) = 0$ y $r(a_1 + a_1^*) + (b_1 + b_1^*) = 0$, es decir, $r^* = r$ para poder ser extraída como factor común y seguir cumpliendo con la condición de no falsedad.

En el **Teorema 1** del paper se demuestra previamente que el algoritmo no devela información sobre el valor de r en ninguna parte, por ende, que r^* sea igual a r es una cuestión aleatoria de probabilidad $P(r = r^*) = \frac{1}{\#GF(251)} = \frac{1}{251}$.

Un detalle muy importante, no mencionado por los autores del paper es que, para entender la eficacia de este algoritmo, la probabilidad de poder engañar la detección de sombras falsas

aplica a un solo bloque, por ende, para poder pasar la detección exitosa de la imagen completa, deberán haber podido acertar de manera azarosa el valor de r_i^* (r^* para el bloque i) en todos los bloques de la imagen. Por lo tanto, a medida que la imagen es más grande, el algoritmo es más efectivo bajo el mismo esquema, ya que al ser los r_i elegidos de forma aleatoria, uniforme e independiente entre si, la probabilidad total de pasar la prueba de falsedad resultaría en $P_{total} = \prod P(x = x_i) = \frac{1}{251^t}$ siendo t la cantidad total de bloques de la imagen.

En conclusion, el algoritmo logra detectar sombras falsas de manera eficaz con la probabilidad mencionada, e incrementa su eficacia a medida que la imagen secreta a compartir es de mayor tamaño.

3. Implicaciones de trabajar en GF(251)

Trabajar en GF(251) acarrea algunos problemas que vamos a explorar en esta sección.

En primer lugar, trabajar con el esquema de imagen secreta compartida en GF(251) implica directamente en que los coeficientes de los polinomios deban ser números enteros positivos en el intervalo $[0, 251]$, es decir, cada pixel de la imagen debe ser representado en estos valores.

En el caso de escala de grises podemos identificar rápidamente un primer problema, los valores 251, 252, 253, 254 y 255 seria convertidos aplicando congruencia módulo 251 a un valor opuesto al original, pasando de ser pixels blancos en la imagen original a pixels negros en la imagen recuperada. Es decir, la imagen recuperada seria distinta a la imagen original.

Esto podría llegar a ser un problema dependiendo de los requerimientos sobre la imagen secreta a compartir, por ejemplo, si se trata de una imagen pequeña, es decir, con pocos pixels, y esta imagen contaría con pixels en los valores mencionados anteriormente, la misma podría mutar demasiado y dejar de ser interpretable por un humano.

4. Posibilidad de guardar secretos de otro tipo

Dado que el algoritmo de imagen secreta compartida que venimos analizando a lo largo de este documento nos permite distribuir en sombras una imagen secreta, podríamos considerar utilizar el mismo para guardar otro tipo archivo como PDFs, archivos binarios ejecutables, otro tipo de imágenes, etc.

Pero dado que estamos trabajando en GF(251) como se explico en la sección anterior, algunos pixels, en este caso, bytes del archivo original, podrían llegar a ser convertidos a su

equivalente en modulo 251. En algunos tipos de archivos o programas, este cambio de representación podría llevar a que el archivo recuperado sea imposible de leer/ejecutar.

Un ejemplo muy claro para entender este problema podría ser, definir un lenguaje de programación ficticio en donde, por ejemplo, el numero 1 en código máquina podría representar un `halt` y el número 252 en código máquina podría representar el comienzo de un `if`, al ser el `if` convertido en un `halt` , un programa en este lenguaje finalizaría en un lugar donde antes ocurría un condicional. Este es solo un ejemplo para comprender la problemática, pero si se analizaran en detenimiento formatos reales de archivos nos daríamos cuenta que como estos problemas vamos a encontrar muchos.

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

El número de sombra en el trabajo práctico de implementación fue convenientemente almacenado en el header `bfReserved1` de las imágenes portadoras en formato BMP.

Otra opción obvia para almacenar este dato es el header `bfReserved2`, otra opción podría ser tomar siempre como mínimo un byte de offset entre headers y pixels para almacenarlo ahí.

Ahora, por el pragmatismo de la implementación planteada en el trabajo practico se decidió que el numero de sombra esta auto contenido dentro de los datos del archivo BMP. Esto no es una condición necesaria ni esta relacionada al algoritmo de imagen secreta compartida, por lo cual, podría almacenarse por fuera de los datos del archivo, ya sea, pensando en un escenario más real, de forma física, es decir, el dueño de la sombra podría tener anotado fisicamente el numero de sombra que posee y proveerlo al programa de recuperación de forma manual. Podría colocarse en un archivo de metadata, etc.

Recapitulando, el numero de sombra puede ser o no almacenado en los datos del archivo. En el trabajo práctico se encuentra almacenado dentro del mismo, pero podría no estar y ser especificado a la hora de ejecutar el programa de recuperación como se observa en el ejemplo debajo.

```
./ss [imagen_a_recuperar].bmp r [k] -1 shadow_1.bmp ... -k shadow_k.bmp
```

6. ¿Qué ocurriría si se permite que r, a0, a1, b0 o b1 sean 0?

Recordemos que en el algoritmo que venimos analizando, a la hora de construir los polinomios, el dealer debe seleccionar un numero r al azar en GF(251), siendo i el número de bloque, que cumpla simultaneamente con:

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

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

A continuación vamos a analizar que sucedería si se permitiera el valor cero para r, a0, a1, b0 o b1. Para ello, debemos tener en cuenta lo explicado en el punto 2, ya que tendrá un impacto directo en la capacidad del algoritmo para detectar fraude.

Para el escenario donde $r_i = 0$ en un bloque , podemos observar lo siguiente:

$$b_{i,0} = b_{i,1} = 0$$

Es decir, para que un adversario pase la prueba de falsedad exitosamente, bastaría con generar sub-sombras con cualquier valor de $f_i^*(j)$, ya que $a_{i,0}$ y $a_{i,1}$ fueron eliminados de la prueba, y luego, podría tomar cualquier g_i^* donde los primero dos coeficientes sean cero para tomar el punto $g_i^*(j)$ para pasar la prueba. Es decir, pasar la prueba de seguridad se vuelve trivial.

El caso donde ambos $a_i = 0$, ambos $b_i = 0$ o $b_i = a_i = 0$, nos da como resultado el mismo escenario anterior.

En conclusion, todos estos escenarios debilitan la eficacia del algoritmo, convirtiendose vulnerable a ataques.

7. Relación entre el método de esteganografía (LSB2 o LSB4), el valor k y el tamaño del secreto y la portadora

En el trabajo práctico de implementación se utiliza un método de esteganografía en donde se oculta la sombra obtenida para cada participante en una imagen portadora. Para ello, se utilizar, según el valor de k, los bits menos significativos de los pixels de la imagen portadora.

En el enunciado de este trabajo se menciona la siguiente regla:

- Si $k = 3$ o $k = 4$, se utilizara LSB4, es decir, los cuatro últimos bits menos significativos de un pixel en la portadora
- Sino, para $k = 5$, $k = 6$, $k = 7$ o $k = 8$ se utilizara LSB2, correspondientes a los últimos 2 bits menos significativos de un pixel en la portadora

Recordemos también, que los secretos que queremos ocultar están formados de la siguiente manera $S_j = v_{1,j}||...||v_{t,j}$, siendo t la cantidad de bloques y que cada $v_{i,j}$ ocupa 2 bytes. Ademas sabemos que en el algoritmo, los bloques se forman dividiendo la imagen original en bloques de $2k-2$ pixels, es decir, $t = \frac{size(Image)}{2k-2}$.

Del párrafo anterior podemos extraer la primer relación entre k y la imagen secreta (I_s a partir de ahora), ya que no contamos con ninguna regla de padding, I_s debe ser divisible por $2k-2$ para poder ser dividida en t bloques completos.

Luego, podemos ver que para almacenar las t sub-sombras $v_{i,j}$ en la imagen portadora (I_p a partir de ahora), vamos a necesitar:

- Si $k = 3$ o $k = 4$, se necesitaran $2 * 2 * t$ bytes en I_p, el primer 2 es por los bytes de $v_{i,j}$ y el segundo 2 es porque estamos trabajando con LSB4, es decir, para almacenar 1 bytes necesitamos 2 bytes en la portadora , luego, desarrollando la ecuación:

$$size(I_p) = 4t$$

$$t = \frac{size(I_s)}{2k-2} = \frac{size(I_p)}{4}$$

$$size(I_p) = 2\frac{size(I_s)}{k-1}$$

Es decir,

$$k = 3 => size(I_p) = size(I_s) \qquad k = 4 => size(I_p) = \frac{2}{3}size(I_s)$$

Lo cual se cumple siempre, ya que en el trabajo que siempre $size(I_s) = size(I_p)$

- Para $k = 5$, $k = 6$, $k = 7$ o $k = 8$, utilizamos LSB2, por ende ahora necesitamos $2 * 4 * t$ bytes en I_p, desarrollando:

$$size(I_p) = 4\frac{size(I_s)}{k-1}$$

$$k = 5 => size(I_p) = size(I_s) \qquad k = 7 => size(I_p) = \frac{2}{3}size(I_s)$$

$$k = 6 => size(I_p) = \frac{4}{5}size(I_s) \qquad k = 8 => size(I_p) = \frac{4}{7}size(I_s)$$

Al igual que en el caso anterior, se cumple siempre por la condición del enunciado $size(I_s) = size(I_p)$.

8. Algoritmo con imágenes en color de 24 bits por pixel

En una imagen BMP de 24 bits por píxel a color, la información de color se codifica generalmente en RGB en cada píxel, es decir, los primeros 8 bits representan el color rojo, los siguientes 8 bits al verde y los últimos 8 bits al azul.

Podemos adaptar nuestro esquema de imagen compartida a este formato con mínimas modificaciones, aunque conlleva las mismas desventajas al emplear el módulo 251. En este caso, consideraríamos que nuestros "secretos" no son píxeles completos, sino partes de un píxel de 24 bits, es decir, un byte correspondiente a cada color.

Por tanto, es posible distribuir una imagen de 24 bits por píxel mediante nuestro algoritmo, entendiendo que cada "píxel secreto" es en realidad un componente de color RGB.

9. Aspectos relativos a la implementación del algoritmo

a. Dificultad de la Implementación

El algoritmo de imagen secreta compartida con detección de sombras falsas es relativamente sencillo de implementar si se cuentan con los conocimientos técnicos y matemáticos necesarios. En mi caso particular, puedo destacar la interpolación de polinomios como la parte mas complicada a realizar.

El mayor problema en la implementación de este tipos de algoritmos es el de detectar errores cuando no se conoce la salida esperada, por eso siempre es recomendable encarar este tipo de proyectos de forma muy modular y probando cada parte por separado para garantizar el correcto funcionamiento de todo el sistema.

b. Posibilidad de extensión o modificación

Como se menciono a lo largo del documento, este algoritmo presenta algunos inconvenientes al estar operando en GF(251), por lo que operar en un escenario que no trate de imágenes que luego van a ser interpretadas por humanos es muy complicado.

Una posible extension seria la mencionada en el punto 8 de este informe, en la cual se menciona una posible implementación del mismo en imágenes a color de 24 bit.

Una posible modificación relacionada al punto 7, es implementar alguna regla de padding para poder utilizar imágenes de cantidad de pixels no divisibles por $2k - 2$.

Por lo mencionado anteriormente, las posibles modificaciones o extensiones de este algoritmo mayormente estarán enfocadas en el procesamiento de los pixels de la imagen secreta y en la distribución de los secretos con técnicas de estaganografía o herramientas similares. No se puede observar a simple vista una extensión viable a otro tipo de archivos o escenarios.

10. Situaciones en las que se utilizaría este algoritmo

Para dar un poco de contexto, en los últimos 2 años me encuentro trabajando para una compañía de NFTs llamada POAP (Proof Of Attendance Protocol). POAP se encarga de distribuir tokens NFTs a personas que asistieron a un evento.

Para una persona poder obtener un POAP es necesario contar con una cuenta de Ethereum ya que el mismo será enviado a la dirección de la cuenta. Una cuenta de Ethereum es básicamente ser dueño, en el contexto de criptografía asimétrica y de forma super resumida, de una clave privada la cual es identificada por el resto de los usuarios en las distintas redes por una clave publica.

¿Que tiene que ver esto con un esquema de secreto compartido?

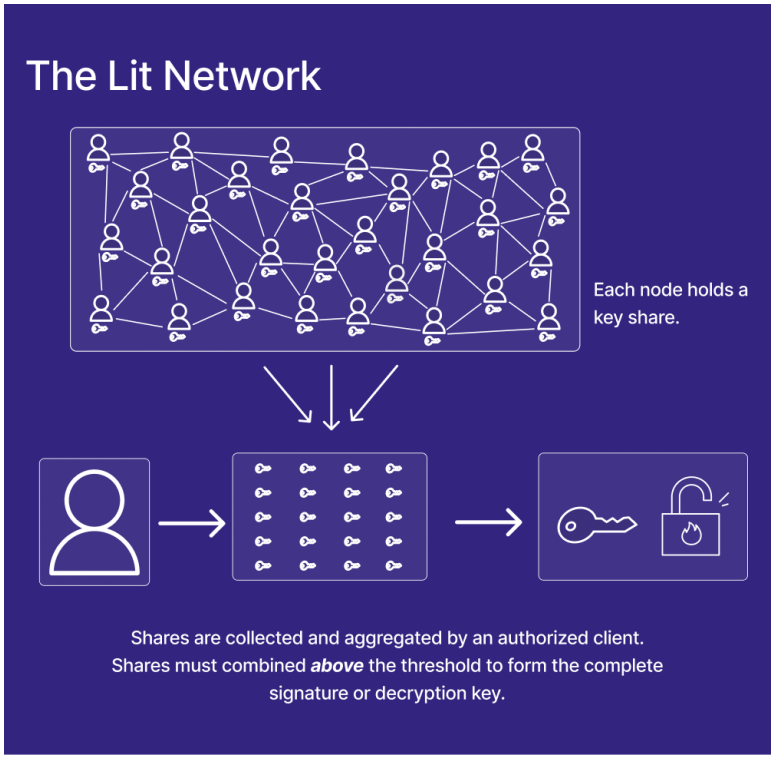
Ethereum, Bitcoin y demás blockchains basadas en que los usuarios posean una clave privada enfrentan un gran problema de trade-off entre seguridad y usabilidad. Actualmente, existen pocas formas, y para nada sencillas, para preservar y administrar de forma segura una clave privada, la más recomendad, es contar con un dispositivo Hardware especifico (Ledger, Trezor, etc) para realizar esta tarea el cual tiene encriptado mediante un PIN la clave privada en el dispositivo, es decir, si uno quiere interactuar con la red de Ethereum con su cuenta necesita tener acceso a este dispositivo físico y ademas conocer el PIN, este método es costoso económicamente y ademas es poco practico. Otra alternativa, es utilizar la clave privada en un dispositivo mobil o en una computadora a través de una billetera virtual, es decir, un programa encargado de custodiar esta clave privada de manera encriptada con una contraseña.

La clave privada en Ethereum puede ser reconstruida mediante 24 o 12 palabras, por lo que una de las mayores formas a la que recurren los usuarios para almacenar un backup de su clave privada es, copiando estas palabras en un papel y almacenandolo fisicamente en un lugar seguro o peor aun, tomando una foto del papel. A simple vista, de esto se desprender muchísimos problemas, el papel podría destruirse, ser robado, extraviarse, etc.

Dado todos estos inconvenientes, el ecosistema Ethereum actualmente están surgiendo muchas alternativas con el objetivo de solucionar todos estos problemas de UX.

Uno de estas soluciones es el de **MPC Wallets** (Multi-Party Computation Wallets), en donde se utilizan esquemas de secretos compartido para distribuir la clave privada de un usuario entre multiples nodos, eliminando la responsabilidad al usuario de tener que almacenar su

clave de forma completa o parcial. A continuación se puede ver un ejemplo de implementación por Lit Protocol, pero cabe destacar que como ellos hay muchos intentando resolver el problema de esta manera.



Además de mejorar la usabilidad, secreto compartido nos habilita a explorar nuevas mejorar de usabilidad como, “Social Recovery”, lo que consiste básicamente en poder recuperar la cuenta cuando perdimos el acceso a nuestra clave privada con la ayuda de otros participantes del esquema.

Para finalizar, dado mi contexto laboral y area de interes, utilizaria esquemas de secreto compartido para solucionar problemas de UX en redes criptográficas como Ethereum.