Criptografía y Seguridad
Secreto compartido

Profesores:
Pablo Abad - Ana Arias Roig - Rodrigo Ramele

Alumnos:
Gonzalo Beade (61223) - Kevin Catino (61643)
Agustina Ortu (61548) - Juan Barmasch (61033)

Grupo: 16

72.44 - Criptografía y Seguridad - ITBA (Instituto Tecnológico de Buenos Aires)

Junio 2023

En este informe nos centraremos en la implementación del algoritmo descrito en el artículo "(k, n) secret image sharing scheme capable of cheating detection" de Yan-Xiao Liu, Qin-Dong Sun y Ching-Nung Yang. Para lograr este objetivo, hemos desarrollado el trabajo en lenguaje C, aprovechando así la oportunidad de familiarizarnos y aplicar técnicas de implementación criptográficas y esteganográficas.

INTRODUCCIÓN

En el siguiente trabajo buscamos familiarizarnos con las nociones de esteganografía y secreto compartido, así como tener un primer acercamiento a la implementación de algoritmos criptográficos.

En este informe exponemos el desarrollo e implementación del algoritmo "(k, n) secret image sharing scheme capable of cheating detection", desarrollado por Yan-Xiao Liu, Qin-Dong Sun y Ching-Nung Yang y discutimos su correctitud, eficiencia y otras decisiones de diseño.

El algoritmo original, escrito por Adi Shamir, busca implementar un esquema de secreto compartido entre n portadores, en el cual a cada uno se le distribuye una parte de un bloque de información común. El bloque de información que se esconde es ilegible para cada una de las partes individualmente, y sólo puede develarse si k de los portadores aportan sus secciones individuales para recuperar el secreto original.

Las implementaciones inocentes del algoritmo asumen que los portadores de las sombras se comportan de manera benigna. Sin embargo, no es raro en la práctica que los agentes sean de naturaleza bizantina y busquen explotar las vulnerabilidades del algoritmo para acaparar información de sombras ajenas y revelar el secreto compartido de manera privada, a costa de los demás portadores. Los autores del artículo original proponen una modificación al algoritmo original que permite bloquear, con cierto grado de posibilidad, estos ataques de "cheating" o engaño.

Las sombras - a los ojos de los portadores - no son diferentes a ruido blanco, son simplemente un buffer de bytes sin sentido alguno. En este informe, complementamos el algoritmo con técnicas esteganográficas, dónde cada sombra se diluye a lo largo de los bits menos significativos de otras imágenes.

CONSIDERACIONES TÉCNICAS

A continuación se presentan algunas decisiones de diseño tomadas en el trabajo práctico.

En primer lugar, se decidió limitar el número de sombras a generar (el valor de n) a 10 dado que se acordó como un número suficiente de sombras en términos de lo que corresponde a la entrega.

Para interpolar valores y generar polinomios, utilizamos la versión optimizada de Lagrange, la cuál usa la versión encajada de un polinomio. Decidimos utilizar esta técnica debido a que el cálculo del polinomio interpolador es de naturaleza recursiva, es un problema que en cada iteración se achica hasta llegar a un caso base. La implementación concreta en nuestro trabajo es iterativa para ahorrar memoria, pero el hecho de que el algoritmo sea recursivo lo hace fácil de entender.

Para leer las imágenes BMP, utilizamos memory-mapped files para cargarlas íntegramente en memoria desde el instante inicial. Es decir, al abrir un archivo BMP con `fopen`, no leemos byte a byte y lo vamos colocando en un buffer con `read`, sino que cargamos la totalidad de los bytes en memoria con con la system call `mmap`. Esto tiene muchas ventajas en términos de eficiencia:

- Incrementamos la performance en I/O pues solo tenemos que ir una única vez a disco por cada archivo BMP. Con la system call `read`, debemos ir a disco cada vez que necesitamos una nueva página del archivo.
- Es más fácil trabajar con los archivos pues podemos acceder a los bytes directamente. Para la parte de esteganografía, podemos pisar bits directamente sobre la imagen mapeada en memoria y después bajarla a disco con mínimo costo implementacional.
- Las imágenes BMP con las que trabajamos en este trabajo no son pesadas, por lo que cargarlas a memoria íntegramente no ocupan un porcentaje significativo de la RAM.

PARTE I - ANÁLISIS DEL ARTÍCULO

Discutir los siguientes aspectos relativos al documento/paper.
a) Organización formal del documento (¿es adecuada? ¿es confusa?)
b) La descripción del algoritmo de distribución y la del algoritmo de recuperación. (¿es clara? ¿es confusa? ¿es detallada? ¿es completa?)
c) La notación utilizada, ¿es clara? ¿cambia a lo largo del documento? ¿hay algún error?

La organización formal del documento es adecuada. El paper sigue una estructura clásica de introducción, desarrollo y conclusiones, y se presenta de manera clara y ordenada.

La descripción del algoritmo de distribución y recuperación es clara, detallada y completa. El paper explica paso a paso cómo se realiza la distribución de la imagen secreta en sombras y cómo se recuperan estas sombras para obtener la imagen original. Además, el paper también describe cómo se detecta el comportamiento fraudulento en el proceso de recuperación.

La notación utilizada es clara y en gran medida coherente a lo largo del documento. Notamos una falta de consistencia en el ejemplo que da el paper respecto al chequeo de "cheating", donde erróneamente utiliza las fórmulas $a_{i0} + r_i \cdot b_{i0} \equiv 0$ y $a_{i1} + r_i \cdot b_{i1} \equiv 0$ en lugar de las explicitadas previamente en el documento ($a_{i0} \cdot r_i + b_{i0} \equiv 0$ y $a_{i1} \cdot r_i + b_{i1} \equiv 0$). En nuestra implementación se considera a esta última como válida, descartando la que se utiliza en el ejemplo del paper.

PARTE II - DETECCIÓN DE ENGAÑOS

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

El documento describe un enfoque para detectar comportamientos fraudulentos en el proceso de recuperación de sombras para obtener la imagen original. El método se basa en esconder un valor compartido entre los bytes k+1 y k+2 de cada bloque en las diferentes sombras.

Lo que el método hace es generar los bytes k+1 y k+2 (b_1 y b_2) de cada bloque a partir de un mismo nonce entero en GF_{251} e información fija del secreto original. Al momento de reconstruir el polinomio, se revisa que los bytes b_0 y b_1 efectivamente sean consistentes y hayan sido generados a partir de la misma componente r.

El escenario de ataque es un atacante maligno que propone una sombra falsa para acaparar la información de sus pares y retenerla solo para él. Si el atacante no provee una versión íntegra, sin adulterar, de su sombra, entonces en la reconstrucción de los polinomios se detectará con altísima probabilidad un intento de cheating, y se parará la ejecución del programa.

La siguiente subrutina permite detectar si hay un intento de cheating. Notar que ya se valida desde antes que aij no sea nulo.

```
static int isCheating(int ai0,
            int ai1, int bi0, int bi1) {
    for (int i=0 ; i< MOD ; i++) {
        if (CONG(ai0*i + bi0) == 0
                && CONG(ai1*i + bi1) == 0) {
            return 0;
        }
    }
    return 1;
}
```

La estrategia desarrollada en el paper no es perfecta. El paper establece que un atacante tiene $\frac{1}{251}$ de probabilidades *por bloque* de salirse con la suya. Es decir, de proponer una sombra falsa, tiene cierto grado de probabilidad de que al reconstruirse los polinomios f_i y

g_i el valor de $ai0, ai1, bi0$ y $bi1$ estén de acuerdo en el valor común r, pero por mera casualidad.

$\frac{1}{251}$ puede parecer un valor no despreciable, pero hay que tener en cuenta de que la probabilidad de que el atacante se salga con la suya es en realidad:

$$\left(\frac{1}{251}\right)^{|bloques|}$$

pues debe acertar al valor de ri para cada uno de los bloques que conforman el secreto. Ahora sí, con esta condición adicional, la probabilidad de éxito para un atacante decae de manera exponencial con el número de bloques y la podemos considerar negligible.

PARTE III - USO DEL MÓDULO 251

> ¿Qué desventajas ofrece trabajar con congruencias módulo 251?

El dominio de cada byte/píxel en 8-bit BMP es $\{0, ..., 255\}$, por lo que los píxeles 251, 252, 253, 254 y 255 (blancos o muy claros) tratados como valores en $GF(251)$ dan la vuelta y se mostrarían en realidad como negros o muy oscuros.

No podemos saber si al recuperar un coeficiente del polinomio - pongamos como ejemplo el 3 - es efectivamente un píxel oscuro de valor 3 o un píxel claro de valor 254 que volvió a empezar el ciclo.

Proponemos tres maneras de solucionar esta limitación.

La primera es multipicar cada byte por $\frac{250}{255}$ para comprimir el dominio de píxeles y que sea compatible con el valor de módulo a utilizar. La profundidad de color será menor, y varios tonos de gris irán a parar a los mismos valores.

La segunda manera es simplemente bajar los píxels más luminosos - con valor mayor a 250 - y truncarlos al valor 250. Devuelta, se pierde profundidad de color, pero solamente en los valores más claros. A diferencia del primer caso, no hay distorsión de los colores en general, sino que solamente se perdería luminosidad en los blancos.

La tercer manera es, simplemente, ignorarlo. Si alguna imágen tuviera bytes entre 251 y 255, solamente serían manchas puntuales en la imagen recuperada y los portadores aún así podrían extraer la gran parte de la información principal.

Las soluciones antedichas son circunstanciales, y dependen de cada caso de uso en particular. Si se desea recuperar al 100% el secreto original, se deben buscar alternativas externas a lo propuestas en el artículo.

Una alternativa que proponemos es prependear un bit en 0 a cada byte y posterioremente sumar 1. Es decir, convertir el byte $b_1b_2b_3b_4b_5b_6b_7b_8$ en $(0b_1b_2b_3b_4b_5b_6b_7 + 1)$ y el bit b_8 interpretarlo dentro de los próximos 8 bits. Esta decisión agranda el tamaño del archivo adicional y de cada sombra en un 12.5% pero garantiza que cada byte x esté comprendido entre $0 \leq x \leq 129 < 251$. Al momento de recuperar la imagen original, se debe restar uno a cada byte y retirar los bits añadidos en la etapa de distribución.

Hay otra desventaja de usar aritmética modular y que no es posible calcular los inversos multiplicativos de todos los números, en particular del 0 y del 251, por lo que hay que agregar validaciones adicionales para estos valores. Nosotros tomamos, por ejemplo para el cálculo de los r_i, a los bytes en 0 como 1.

> Con este método, ¿se podrían guardar secretos de cualquier tipo (imágenes, pdf, ejecutables). ¿por qué?

Los secretos en sí pueden ser cualquier buffer de bytes arbitrarios, siempre teniendo en cuenta que puede ocurrir que se pierda información en el proceso. En el caso de una foto, la pérdida de información simplemente hace que algunos píxeles cambien de color, pero el archivo sigue siendo estructuralmente correcto. Si queremos que el secreto sea un archivo PDF, un ejecutable, etc., necesitamos garantizar que la información que recuperamos sea *exactamente igual* a la que escondimos. Si podemos dar garantías de esto, entonces cualquier archivo puede ser tomado como secreto.

Por otro lado, no podemos ocultar secretos en archivos de cualquier tipo. Es decir, para la parte de esteganografía del trabajo práctico, no podemos utilizar binarios arbitrarios pues no podemos saber, en general, el significado de esos bits. Por ejemplo, modificar bytes de un ejecutable corrompen el programa con altísima probabilidad, ya que puede traducirse a instrucciones sin sentido, direcciones de memoria prohibídas o sin información o cambiar el valor de variables globales a basura, entre otras cosas.

Existen otras alternativas para esconder secretos en archivos binarios, pero no de manera esteganográfica. Tomemos por ejemplo un ejecutable compilado con GCC. Una vez compilado un ejecutable, le podemos agregar secciones customizadas que contengan informa-

ción secreta que no afectan el flujo normal del programa.

```
#!/bin/bash

echo 'este es un secreto' > hola

# creamos la sección llamada SECRET
objcopy main --add-section .SECRET=hola

# recuperamos la sección llamada SECRET
objcopy --dump-section .SECRET=mysection main

# que contiene la seccion?
cat mysection

> este es un secreto
```

El código anterior muestra cómo se puede guardar un buffer de bytes arbitrario como una sección en un archivo ejecutable compilado con GCC. Podemos utilizar esta idea para ocultar en secciones de un ejecutable sombras generadas por el algoritmo que estamos estudiando.

PARTE IV - USO DE SOMBRAS

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

Se podría guardar en un archivo separado que esté protegido por medidas de seguridad adicionales, como encripción o autenticación. Se podría guardar un hash de la sombra, seguido por el número de la misma. Así, se puede consultar el archivo para recuperar el número de sombra.

Otra opción que proponemos es esconder el número de sombra del propio buffer de bytes que se esconden esteganográficamente. Por ejemplo, que el primer byte tenga información de la sombra en sí y que los datos de la misma empiecen a partir del siguiente byte.

No es necesario que el número de sombra se mantenga oculto criptográficamente, debido a que es fácilmente obtenible por fuerza bruta. Supongamos que tenemos n sombras pero no tenemos su `id` correspondiente. Como ese id pertenece a $GF(251)$, hay exactamente 251 valores que puede tomar. Asumiendo que no se repite el número de sombra (que sería lo correcto), basta con hacer como mucho $251 \times 250 \times ... \times (252 - n)$ intentos, un número acotado y completamente computable por fuerza bruta.

PARTE V - CASOS BORDE

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

Analicemos caso por caso por qué esos valores no pueden ser nulos (o congruentes a 0 en $GF(251)$. Recordemos que las ecuaciones para el cálculo de $b0$ y $b1$ son:

$$a0 \cdot r + b0 = 0 (mod\ 251)$$

$$a1 \cdot r + b1 = 0 (mod\ 251)$$

Notemos que si $a0$, $a1$ y r no son 0, $b0$ y $b1$ no pueden ser cero, pues 251 es primo. Lo que debemos evitar es que $b0$ y $b1$ lleguen a valer 0 en una primera instancia, debido a que sino solamente obtendríamos ecuaciones triviales de resolver $ai \cdot r = 0$.

Supongamos primero que r pudiese ser 0. Esto haría que $b0$ y $b1$ también sean forzados a valer 0 al momento de calcular el polinomio g. Momentos más tarde, al detectar un posible cheating, lo que ocurriría es que cualquier valor para $a0$ y $a1$ resolvería las ecuaciones en simultáneo y sería trivial esquivar el chequeo para ese bloque. Reduce el nivel de seguridad de manera innecesaria. Lo mismo ocurriría si $a0$ y $a1$ fuesen simultáneamente 0.

Supongamos ahora que $a0$ o $a1$ pudieran ser 0. Eso anularía una (o dos) de las ecuaciones y la seguridad se reduciría en proponer un valor arbitrario r que satisfaga una ecuación, lo cuál es trivial.

Los valores no pueden ser cero para proveer un esquema de detección de engaño correcto - que ande en todos los casos - y seguro - que no aumente la posibilidad de éxito de un atacante.

PARTE VI - ESTEGANOGRAFÍA

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

El método de esteganografía LSB (Least Significant Bit) se utiliza para ocultar información dentro de una imagen o archivo de audio sin que sea detectable a simple vista o audición.

El método `LSB2` utiliza los dos bits menos significativos de cada byte en la imagen portadora para ocultar un

cuarto de byte del secreto y por lo tanto se necesita una imagen portadora con al menos 4 veces el tamaño del secreto para ocultar todos los bits del mismo.

El método `LSB4` utiliza los cuatro bits menos significativos para ocultar un nibble de cada byte del secreto y, por lo tanto, se necesita una imagen portadora con al menos 2 veces el tamaño del secreto para ocultar todos los bits del mismo.

Por esta razón es que la consigna nos pide que utilicemos con $k \in \{3, 4\}$ LSB4 y con $k \in \{5, 6, 7, 8\}$, LSB2. Recordemos que las imagenes portadoras son de igual tamaño que la imagen secreta. A continuación, demostramos la necesidad numérica de utilizar LSB4 con los primeros dos valores de k.

$$3 \leq k \leq 4$$

$$2 \leq k - 1 \leq 3$$

$$\frac{1}{2} \geq \frac{1}{k-1} \geq \frac{1}{3}$$

$$\frac{1}{2} \geq \frac{1}{k-1} \geq \frac{1}{3}$$

$$\frac{|secret|}{2} \geq \frac{|secret|}{k-1} \geq \frac{|secret|}{3}$$

$$\frac{|secret|}{2} \geq |shadow| \geq \frac{|secret|}{3} > \frac{|secret|}{4}$$

Notemos que el tamaño de la sombra $|shadow|$ es mayor estricto que la cantidad de bytes disponibles para ocultar la imagen. En cambio, con los demás valores de k esto no ocurre y el tamaño final de la imagen es más chico que un cuarto de la imagen original.

$$5 \leq k \leq 8$$

$$4 \leq k - 1 \leq 7$$

$$\frac{1}{4} \geq \frac{1}{k-1} \geq \frac{1}{7}$$

$$\frac{|secret|}{4} \geq \frac{|secret|}{k-1} \geq \frac{|secret|}{7}$$

$$\frac{|secret|}{4} \geq |shadow| \geq \frac{|secret|}{7}$$

Se suele preferir el uso de `LSB2` por sobre `LSB4` cuando el tamaño de las portadoras no es un problema debido a que la modificación en las imágenes es mucho menos perceptible.

En general, cuanto mayor sea el valor k (es decir, más sombras sean necesarias para recuperar la imagen original), mayor será la cantidad de imágenes portadoras requeridas. Si el valor de k es lo suficientemente grande (k=9, por ejemplo sería suficiente) se podrían usar mejores técnicas esteganográficas como LSB1.

Además, cuanto mayor sea el tamaño del secreto a ocultar, mayor será la cantidad de espacio necesario en las imágenes portadoras y por lo tanto mayores serán las imágenes requeridas.

Vale aclarar que en nuestra implementación asumimos que el tamaño de la imagen a particionar debe ser divisible por $k - 1$. Por ejemplo, si la imagen secreta es de 300x300 y utilizamos $k = 8$, entonces no será posible distribuir la imagen. En futuras extensiones del trabajo, proponemos el uso de padding del tipo *Des Pad*, es decir, agregar un bit 1 seguido de tantos 0s sea necesario para que se acomode a los requerimientos de multiplicidad.

PARTE VII - VARIANTES DEL ALGORITMO

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

Es natural la extensión del algoritmo para imágenes de tres canales de color. Como el secreto, a los ojos del algoritmo, es un búfer de bytes arbitrarios, entonces una imagen blanco y negro de tamaño $3n$ o una imagen a color de tamaño n son indistinguibles, siempre respetando las restricciones de tamaño.

El problema con los bytes en 0 o mayores a 251 sigue existiendo, y generará *píxeles* incorrectos en la imagen final con alrededor del triple de probabilidad. Notar que basta que un único de los canales de un píxel se modifique para que el píxel cambie de color. No es mayor inconveniente, pues la mayor parte de la información se seguirá manteniendo en la recuperación del secreto.

PARTE VIII - ANÁLISIS DE LA IMPLEMENTACIÓN

Discutir los siguientes aspectos relativos al algoritmo implementado:
a. Facilidad de implementación
b. Posibilidad de extender el algoritmo o modificarlo.

El algoritmo de esteganografía LSB es relativamente fácil de implementar, ya que se basa en la modificación de los bits menos significativos de los píxeles en una imagen portadora. El método LSB2 y LSB4 son variantes simples del algoritmo básico, y su implementación también es sencilla. Además, el uso de bibliotecas y herramientas especializadas puede simplificar aún más la implementación del algoritmo.

Este algoritmo es altamente modificable y extensible. Por ejemplo, se pueden utilizar diferentes métodos para seleccionar los píxeles en los que se oculta la información (en lugar de simplemente utilizar los bits menos significativos), o se pueden utilizar diferentes técnicas para codificar la información oculta. Ya mencionamos a lo largo del trabajo múltiples maneras de extender o remixear el algoritmo original, tanto la parte de esteganografía como la parte de secreto compartido (imágenes a color, archivos de audio, esconder como secciones de ejecutables con `objdump`, etc.).

Es importante tener en cuenta que cualquier modificación o extensión del algoritmo debe ser cuidadosamente evaluada para garantizar que no comprometa la seguridad o facilite la detección por parte de herramientas especializadas. Además, siempre debemos revisar la integridad final de los datos, debido a que si usamos la versión original del algoritmo alguno bytes pueden verse modificados en la recuperación final de las sombras.

¿En qué situaciones aplicarían este tipo de algoritmos?

Hay dos algoritmos que se están utilizando en el trabajo práctico: de secreto compartido y de esteganografía.

Los algoritmos de secreto compartido se utilizan cuándo se desea esconder información y que esta no pueda ser revelada a menos que se comuniquen k de las n partes portadora de sombras. Siempre que haya un secreto que necesita múltiples partes para poder revelarse, los algoritmos de secreto compartido tienen un rol especial. Un caso de uso de la vida real podría ser esconder una fórmula secreta para una bebida Cola: solamente se revela cuándo varias personas se sincronizan para desencriptarla. Otro uso de secreto compartido podría ser Seguridad Nacional, dónde solamente se autoriza el despliegue de cierto armamento o táctica si k de los n generales lo permiten.

Los algoritmos de esteganografía son útiles en cualquier situación en la que sea necesario ocultar información dentro del contenido digital sin llamar la atención sobre su presencia. Algunos ejemplos concretos podrían ser:

- Almacenar alguna clave criptográfica dentro de una carpeta de imágenes sin llamar la atención. Sólo basta con conocer cuál es la imagen que contiene el secreto.
- Enviar datos confidenciales en situaciones de urgencia donde es necesario hacerlo a través de un medio inseguro y otras opciones criptográficas no se encuentran disponibles.
- Agregar una marca de agua digital a ciertos documentos (como podrían ser imágenes) con información acerca del dueño del mismo y otros datos adicionales.