UNIVERSIDAD AGRARIA DE LA HABANA

"Fructuoso Rodríguez Pérez"

FACULTAD DE CIENCIAS TÉCNICAS

DEPARTAMENTO DE CIENCIAS BÁSICAS

ALGORITMO DE MARCAS DE AGUA PARA SOFTWARE

Tesis presentada en opción al título de Master en Biomatemática

Autor**:** Ing. Yoandry Milian Nuñez

Mayabeque, 2016

UNIVERSIDAD AGRARIA DE LA HABANA

"Fructuoso Rodríguez Pérez"

FACULTAD DE CIENCIAS TÉCNICAS

DEPARTAMENTO DE CIENCIAS BÁSICAS

ALGORITMO DE MARCAS DE AGUA PARA SOFTWARE

Tesis presentada en opción al título de Master en Biomatemática

Autor**:** Yoandry Milian Nuñez

.

Tutores:

Dr. C. Nelson Nápoles Hernández

Mayabeque, 2016

AGRADECIMIENTOS

A mi tutor Dr.C. Nelson Nápoles por su paciencia, ayuda y dedicación durante el desarrollo de esta investigación, brindando su apoyado en todos los momentos cuando pensaba que no terminaba la investigación en tiempo. A la profesora MSc. Yolanda Jiménez. Le agradezco especialmente a mis padres por su dedicación, amor, educación, ejemplo y apoyo, que siempre me han dado y de lo que estaré agradecido de por vida. Gracias por estar siempre conmigo. A mi familia que nunca ha dejado de guiarme y apoyarme durante el transcurso de esta investigación. A mi novia la cual supo comprenderme y apoyarme en esta recta final, a la que quiero mucho. A mis amistades dentro y fuera de la universidad por brindarme apoyo incondicional cuando lo necesité, en especial a mi amigo Orestico por haber estado siempre ahí cuando lo necesité. A mi madrina Nieves que siempre me apoyo y me daba aliento. A todos aquellos que de una forma u otra han ayudado a mi formación como profesional y como persona. A mis profesores por estar pendiente de mí; a la profesora Dr.C. Lucia Fernández por estar siempre diciéndome que si se podía y apoyándome en los momentos que la necesité. A mis compañeros de trabajo por poder contar con ellos siempre, Alina por estar siempre preocupadas por mí y dándome aliento cuando creía que no la terminaba. En general a todos: GRACIAS.

DEDICATORIA

A mis padres, por creer en mí una vez más y por sus consejos que han hecho de mí cada día una persona mejor, por su entrega y sacrificio, por su ejemplo y amor que siempre me han proporcionado. A mi novia por su amor y paciencia. En general a toda mi familia por haber creído en mí.

RESUMEN

La presente investigación se lleva a cabo en la Carrera de Informática de la Universidad Agraria de La Habana (UNAH) “Fructuoso Rodríguez Pérez”, con el propósito de utilizar un algoritmo que permita la protección de los productos producidos en dicha carrera, donde se destaca la técnica de marcas de agua para software (*software watermarking*) como vía de solución. La misma permite empotrar un mensaje en un *software* para demostrar su autoría, y por tanto, la autenticidad del mismo, esta técnica de protección está sometida a varios ataques. Con el algoritmo propuesto se puede proteger la propiedad intelectual del *software* producido en la Carrera de Informática de la UNAH, lo cual puede ser extendido a otras áreas de la universidad, así como a otras instituciones del país.

Palabras Claves: *Software watermarking*, algoritmo.

ABSTRACT

The present research is done in the Career of Informatics of the Agrarian University of Havana (UNAH) Fructuoso Rodríguez Pérez, with the objective to use an algorithm that allow the protection of the products produced in this career, where the technique of watermarks for software (software watermarking) like way of solution stands out. The same one allows embedding a message in a software to demonstrate its authorship and therefore, the authenticity of the same, this protective technique is to undergo to several attacks. With the proposed algorithm can be protect the intellectual property of the software produced in the career of Informatics of the UNAH, which can be extended to other areas of the university, as well as to other institutions of the country.

Keyword: Software watermarking, algorithm

INDICE

Índice de Tablas y Figuras

INTRODUCCIÓN

Desde el surgimiento de la informática la protección de los contenidos digitales ha sido difícil de resolver, la copia ilegal de contenidos digitales se ha convertido en una práctica habitual. Cuando el contenido digital es un programa (*software)*, las facilidades de copia, almacenamiento y el intercambio propician la piratería. Por citar un ejemplo, en el año 2013, según el reporte de piratería de *software* brindado por la Alianza de Negocios de *Software* (*Business Software Alliance* – BSA), las pérdidas monetarias por concepto de piratería de *software* ascienden a billones de dólares.

Para evitar la piratería de *software* es necesario garantizar ciertos atributos de seguridad, los cuales se destacan: la privacidad, la integridad, la autenticidad y la disponibilidad. Estos atributos son garantizados en cierta medida mediante técnicas de protección de *software*. Entre las que destacan: ofuscación, cifrado de código, auto-verificación de integridad, diversidad de código, marcas de aguas para *software*, entre otros (Nagra, 2005).

La piratería de *software* habitual es la más extendida, y se basa en la obtención gratuita de copias de una determinada aplicación Moruno (2008). Se produce por la compartición de archivos, ya que las copias de las aplicaciones son obtenidas de otros usuarios de forma gratuita, no hay un ánimo de lucro presente.

Por el contrario, la piratería con fines comerciales, se usan técnicas de ingeniería inversa para obtener información vital de un programa, y obtener un beneficio económico (BSA, 2011).

Este beneficio económico se puede lograr mediante la venta de una aplicación equivalente a la original, o mediante el uso de partes del código de una aplicación. El objetivo de los atacantes es eliminar la marca de agua (*watermark*) o imposibilitar su reconocimiento.

Mientras que el objetivo de las técnicas de marcas de agua para *software* (*software watermarking)* es resistir el mayor número de ataques posible. No existe ningún *watermark* infalible, porque contando con tiempo y recursos suficientes un *watermark* siempre puede ser corrompido. El éxito de eliminar un *watermark* estará en la propia resistencia que ofrezca el mismo en cuanto a recursos y tiempo.

La presente investigación se centra en abordar las técnicas que garantizan la autenticidad de los sistemas informáticos, destacándose la técnica de marcas de agua para *software* (*software watermarking*) como vía de solución.

Las marcas de agua para *software* permiten empotrar un mensaje en un *software* para demostrar su autoría y, por tanto, la autenticidad del mismo. A estas técnicas de protección se le pueden hacer varios ataques los cuales pueden ser: de sustracción, deformación, adicción y confabulación (Collberg, 2002).

Estas técnicas se amparan en la Ley de la Propiedad Intelectual[1], de esta forma se disuade a usuarios malintencionados de hacer uso de un *software* no autorizado por sus propietarios y utilizarlos con fines de lucro comercial.

Esta técnica permite certificar en caso de litigio, la propiedad del producto y para ello se inserta una marca de derecho de copia (*copyright*)[2] que pueda resistir posibles ataques e imposibilitar su posterior extracción. Un ejemplo de ello, se evidencia a la hora de proteger un *software* utilizando las técnicas de marcas de aguas para *software* que identifican a un fabricante como propietario legítimo del *software*.

[1] La ley de Propiedad Intelectual: el reconocimiento de un derecho particular en favor de un autor u otros titulares de derechos, sobre las obras del intelecto humano.

[2] Expresa los derechos de autor, cuya intención es proteger la propiedad intelectual de un uso ilegal y evitar que se alegue dichos derechos a una persona o institución que no le corresponda.

Una violación del *copyright* trae como consecuencia pérdidas económicas a las fábricas de *software* y por consiguiente un deterioro de la economía del país. Por citar un ejemplo, estudios realizados en Argentina por *Business Software Alliance* (BSA) y la Corporación Internacional de Datos (*Internacional Data Corporation* - IDC) señalan los beneficios económicos de reducir la piratería de *software* (BSA, 2011).

Reducir la piratería de Argentina en un 10 % en dos años en vez de 4 aumentaría el impacto económico en un 35%. La Empresa antes mencionada hizo un estudio a escala mundial en el cual se expresa que, a nivel global, los datos muestran que reducir la piratería de *software* en un 10 % en los próximos 4 años producirá US$142 mil millones en nuevas actividades económicas en los 42 países estudiados, con más de 80% acumulable para las industrias locales.

La reducción también crearía cerca de 500 000 puestos de trabajo de alta tecnología y generaría alrededor de US$32 mil millones en ingresos por impuestos a nivel mundial. Concentrar de manera anticipada las ganancias reduciendo la piratería en el 10% en dos años capitaliza los beneficios económicos en un 36%, produciendo US$193 mil millones en nuevas actividades económicas generando US $43 mil millones en ingresos por impuestos (BSA, 2011).

En Cuba la Ley 14 que es la referida a la propiedad intelectual, no contempla de forma explícita lo vinculado a las TICs. El país viene desarrollando *softwares* que son comercializados a otros países. Es de destacar que referente al tema, en Cuba solo se ha llevado a cabo un estudio limitado a una revisión bibliográfica por parte de la Universidad de Oriente sobre las técnicas de marcas de agua para *software*.

Según el criterio de Hernández relacionado con la propiedad intelectual es importante tener en cuenta que:

"En el contexto de la sociedad actual, la actividad universitaria está vinculada a una gestión apropiada del conocimiento en el sentido de los modos de producción intelectual y científico; la difusión y transmisión de conocimientos y saberes; la gestión de la propiedad intelectual e industrial; la circulación de los recursos humanos con elevados niveles de formación y experiencia; y el uso de inventos y su comercialización" (Hernández, 2013: 14).

En el contexto de la Universidad Agraria de La Habana (UNAH), a pesar de contar con una Política Científica la cual incluye un acápite de propiedad intelectual, en su mayoría no se gestionan los resultados investigativos ni se protegen los derechos intelectuales sobre los mismos. Algunos resultados de la universidad han sido protegidos por las modalidades correspondientes de la propiedad industrial, sin embargo, debido a la falta de organización y a la ausencia de conocimiento sobre el tema, se han perdido oportunidades y no se ha podido aprovechar al máximo el potencial innovador de la UNAH (Taboada, 2010).

A las universidades les ha sido asignada históricamente la misión de crear y difundir el conocimiento básico que se genera en el seno universitario. Sin embargo, en los tiempos actuales, la misión de las altas casas de estudio ha cambiado. La sociedad requiere no sólo del conocimiento básico, sino que es imprescindible el conocimiento aplicado (Hernández, 2013).

En las universidades cubanas donde existen carreras que tributan a la producción de *software* deben tener algoritmos de protección lo cual evitaría el uso del producto con el fin de obtener un lucro comercial. En la Carrera de Ingeniería Informática de la UNAH, se constató a través de consultas a directivos y profesores, que los s*oftwares* producidos en su mayoría no disponen de técnicas de protección que garantice su derecho de autoría, para neutralizar la piratería.

Teniendo en cuenta estos antecedentes referidos a la protección del *software* que garantice su derecho de autoría se define como:

Problema científico: Las aplicaciones producidas en la Carrera de Ingeniería Informática de la Universidad Agraria de La Habana no contienen un algoritmo de marca de agua para *software* que los proteja de la piratería con fines de lucro comercial.

El **objeto de estudio** en esta investigación son las marcas de agua para *software*; cuyo **campo de acción** es las marcas de agua de para *software* en la Carrera de Ingeniería Informática Universidad Agraria de La Habana.

El Objetivo General: Desarrollar un algoritmo de marcas de aguas para *software* que permita la protección de las aplicaciones producidas en la Carrera de Ingeniería Informática de la Universidad Agraria de La Habana contra la piratería con fines de lucro comercial.

Objetivos Específicos:

1. Revisar los fundamentos teóricos y metodológicos relacionados con las marcas de agua para *software*.
2. Diagnosticar las marcas de agua para *software* en la Carrera de Ingeniería Informática en la Universidad Agraria de La Habana.
3. Propuesta e implementación de un algoritmo de marcas de agua para *software* el cual se utilizará en la Carrera de Informática de la Universidad Agraria de La Habana.

Hipótesis

Con un algoritmo de marca de agua para *software* es posible la protección de las aplicaciones producidas en la Carrera de Ingeniería Informática de la Universidad Agraria de La Habana evitando la piratería con fines de lucro comercial.

Dentro los métodos de investigación utilizados están los métodos teóricos y empíricos:

MÉTODOS TEÓRICOS

Histórico-Lógico:

Permitió conocer cuál ha sido la evolución y desarrollo de las marcas de aguas para *software* así como los tipos de ataques teniendo en cuenta los criterios de autores de la comunidad científica nacional e internacional.

Análisis y Síntesis:

Con el objetivo de separar el objeto de la investigación en cada una de sus partes. Descubrir sus interrelaciones. Además de poder realizar un análisis de la información obtenida en las fuentes bibliográficas, así como la elaboración final de la investigación.

Inducción-deducción:

Permitió hacer un análisis de los conceptos teóricos expuestos por diferentes autores relacionados con las marcas de agua para *software*, estableciendo en su análisis la relación entre lo singular, lo particular y lo general.

Tránsito de lo abstracto a lo concreto:

Permitió evaluar los cambios que se van dando en el proceso de investigación y cómo se reflejan estos en el tema a investigar.

MÉTODOS EMPÍRICOS

Encuestas:

Realizadas a profesores y estudiantes de la Carrera de Ingeniería Informática de la UNAH para conocer sus criterios acerca de las marcas de agua para *software* y la importancia de la protección de los mismos.

Análisis documental:

Permitió realizar un análisis de los documentos encontrados durante el proceso de búsquedas relacionados con el tema de investigación en la revisión bibliográfica como parte de la fundamentación teórica de la investigación y en el análisis de las tesis de diploma presentadas por estudiantes de la Carrera de Informática de la UNAH.

Novedad científica:

La novedad de la investigación radica en el diseño de un algoritmo de marcas de agua para *software* que contribuya a la protección de los mismos en la Carrera de Informática de la Universidad Agraria de La Habana, con el objetivo de neutralizar la piratería con fines de lucro comercial. Dicha propuesta pudiera ser generalizada a las empresas e instituciones, nacionales que produzcan *software*.

Significación práctica

Las empresas y centros en los que se produzcan *software* contarán con un algoritmo de marca de agua para *software* que permitirá neutralizar la piratería con fines de lucro comercial.

Estructura de la tesis

La tesis está estructurada en introducción y dos capítulos, con sus respectivos epígrafes; de los cuales a continuación se ofrece un resumen del contenido abordado en cada uno:

Capítulo 1. Fundamentos Teóricos y Metodológicos Relacionados con las marcas de agua para *software*, este capítulo sintetiza los criterios de autores de la comunidad científica nacional e internacional, permitiendo fundamentar la propuesta.

Capítulo 2. Se realiza la caracterización de la Carrera de Ingeniería Informática en la UNAH, se exponen los resultados del análisis documental y el diagnóstico aplicado, además de la propuesta e implementación del algoritmo de *software watermark*ing.

El trabajo cuenta además con: conclusiones, recomendaciones, bibliografía y anexos.

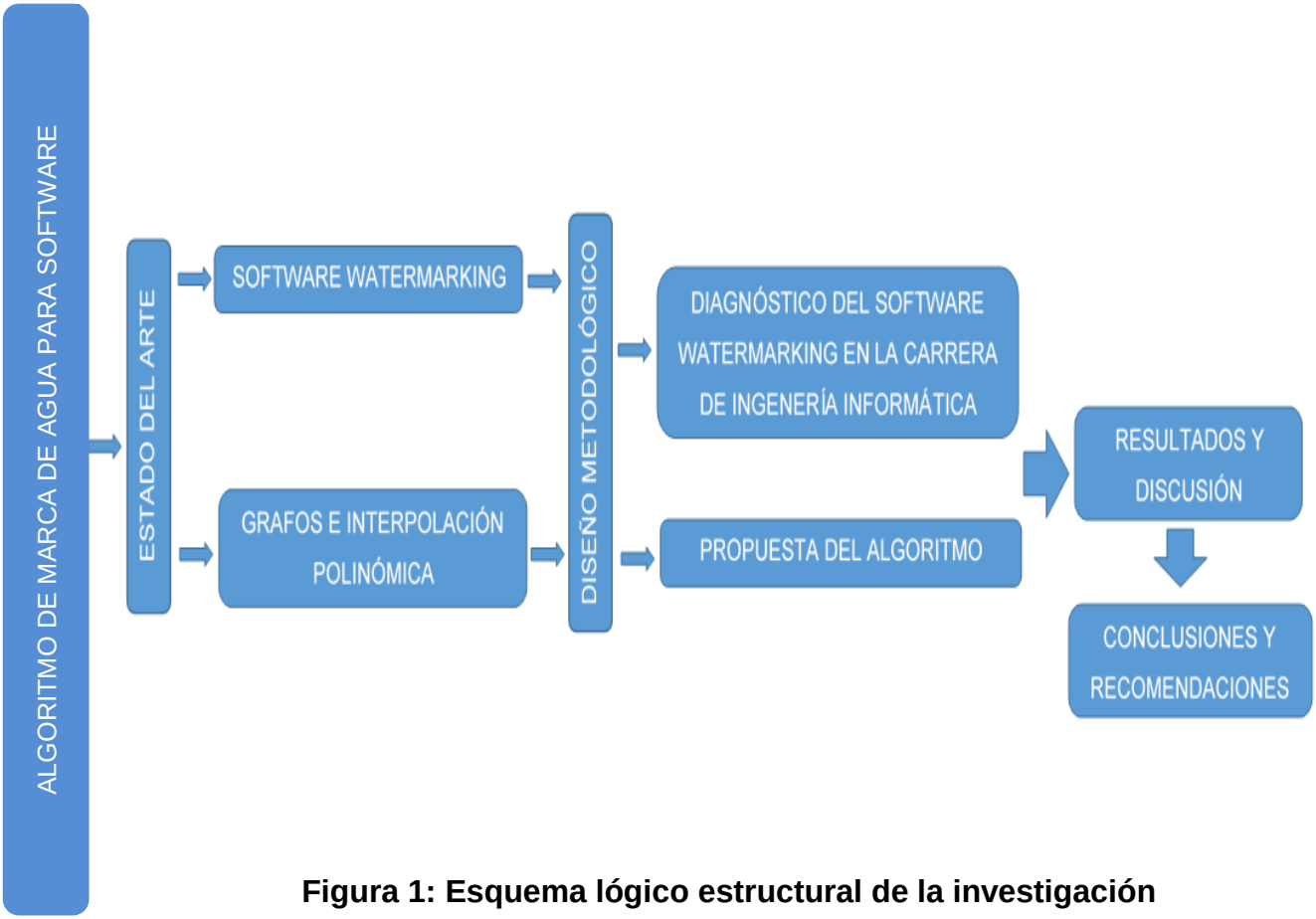

Figura 1: Esquema lógico estructural de la investigación

Fuente: Elaboración propia

CAPÍTULO 1. FUNDAMENTOS TEÓRICOS Y METODOLÓGICOS RELACIONADOS CON LAS MARCAS DE AGUA PARA *SOFTWARE*

El presente capítulo contempla lo relacionado con las marcas de agua para *software*, teniendo en cuenta los criterios de autores de la comunidad científica nacional e internacional, lo que permitirá fundamentar la propuesta y discutir teóricamente los resultados que se obtengan del diagnóstico. En el epígrafe 1.1 se explica de manera general como es llevada a cabo la protección de *software*, así como los tipos de ataques que se pueden realizar a los mismos incluyendo las medidas contra dichos ataques. Seguidamente en el 1.2 se centra en las marcas de agua para *software* dado que es el objetivo principal que motivó la presente investigación. Consecuentemente, se enumeran los principales elementos que conforman este tipo de técnica de protección de *software*. En el 1.3 y 1.4 se abordan las diferentes clasificaciones de las marcas de agua para *software*. En el 1.5 se enuncian las diferencias entre la estenografía, criptografía y las marcas de agua para *software* lo cual sienta las bases que permitan sugerir el algoritmo. Posteriormente, en el 1.6 se explica en que consiste la compartición de secretos lo que servirá de núcleo en el algoritmo propuesto. Por último, en el 1.7 se brindan los conceptos matemáticos necesarios para el algoritmo propuesto.

1.1 Protección de *software*

En la protección de un *software* del ataque de un *host* malicioso, no existe ninguna defensa infalible. Una vez infectada la terminal, se puede usar cualquier técnica para extraer los datos que se requieran, o violar su integridad. En realidad, los únicos factores que limitan esta vulnerabilidad son los recursos que puede invertir el *host* para analizar el código del programa.

Si un usuario malintencionado dispone de tiempo y recursos computacionales el cual le permita obtener el código de una determinada aplicación para modificarlo y luego recompilarlo, logrando así crear su propia versión del mismo y distribuirla bajo otro nombre. Esto generaría perdidas monetarias a la empresa la cual fue la que creó el *software*. Por consiguiente, la alternativa para la protección del *software* radica, en

encontrar esquemas de protección que resulten excesivamente costosos de romper, en términos de recursos.

1.1.1 Tipos de ataques

Se definen tres tipos de ataques a la propiedad intelectual contenida en un programa, La ingeniería inversa, la piratería, y el *tampering* de *software*. Una potente defensa frente a la ingeniería inversa es la ofuscación de código, un proceso que hace que un programa sea ininteligible pero funcional. Una defensa ante la piratería de *software* son las marcas de agua para *software*, que permite determinar la autoría de un programa. Y finalmente, una defensa ante el *tampering* es el llamado *tamper-proofing*, de tal modo que aquellas modificaciones no autorizadas dan como resultado un código no funcional (Collberg and Thomborson, 2002).

Ingeniería inversa: Es el proceso de extracción del conocimiento o del diseño de un programa desde cualquier cosa hecha por el hombre. El proceso consiste en analizar un programa con el objetivo de obtener conocimiento sobre el funcionamiento del mismo sin disponer del código fuente original. Una vez usada la ingeniería inversa para obtener esta información, se aprovecharía para modificar el código y usarlo en futuras aplicaciones con el mismo u otro propósito (Eilam, 2005).

***Tampering*:** consiste en realizar modificaciones no autorizadas en un programa. El objetivo de estas modificaciones puede ser la obtención de información restringida, o la supresión de mecanismos de control de licencias de uso. La mayoría de los ataques están sucedidos de ataques de ingeniería inversa ya que se realiza un análisis de cómo funciona para luego centrarse el módulo de interés o en algunos casos para suprimir el *watermark*.

Piratería de *software*: es la copia, distribución, utilización o elaboración ilegal de programas informáticos que constituyen actos de agresión contra la propiedad intelectual y se considera como actos punibles en los códigos penales.

La piratería de *software* se puede clasificar en dos grupos: la piratería sin ánimo de lucro, la cual es producida por el intercambio del *software* con el fin de obtener copias gratuitas u organizaciones que no reportan el número real de instalaciones del material. El otro grupo es la piratería con fines de lucro, en la que los principales atributos que persigue es la falsificación, duplicación y distribución a gran escala de *software* copiado ilegalmente provocando el enriquecimiento ilícito del que la efectúa y promueve el deterioro de la economía del estado, y lo que aún es más grave: incita a su incorporación nuevos talentos y privatiza al propietario de los derechos *copyright*.

1.1.2 Medidas contra los Ataques

Ofuscación: es la técnica de transformar el código en otra menos legible y de difícil compresión al ser humano, pero manteniendo su funcionalidad, de esta forma se modifica el código para esconder su propósito original y ahogar al agresor de información. Según nivel de complejidad añadida por una ofuscación (transformación) es típicamente llamada potencia, y puede ser medida usando métrica convencional de complejidad del *software*, analizando cuántos predicados el programa contiene y la profundidad de anidar en una secuencia particular de código (Eilam, 2005).

Un deofuscador es un programa que implementa algoritmos de análisis de flujo de datos diversos en un programa ofuscado que a veces posibilita metafóricamente separar el trigo de la paja desmenuzada y automáticamente elimina todas las instrucciones irrelevantes y restaura la estructura original del código. Crear transformaciones de ofuscación que sea resistente a la deofuscación es un reto principal y es la meta primaria de muchos ofuscadores. Finalmente, la ofuscación tendría un costo asociado.

Esto puede ser producto a que el código sea más grande, por consiguiente, el tiempo de ejecución es más lento, lo que conlleva al aumento del consumo de rutina de memoria. Algunas transformaciones del código no merecen cualquier clase de costos de la rutina de memoria, porque involucran una simple reorganización del programa que es transparente para la máquina, excepto los programas menos legibles para el humano.

La ofuscación automática es obviamente mucho más efectiva porque puede ofuscar el programa entero y no simplemente las partes pequeñas de él. Es importante considerar que la ofuscación automática es típicamente realizada después de que el programa se ha compilado y no como es el caso cuando la ofuscación es realizada manualmente.

Tamper-proofing: Es la técnica que se utiliza cuando nos interesa impedir que un usuario ejecute el programa si éste ha sido alterado. Para lo que se definen algunas situaciones en las que un programa P no debería poder ejecutarse (Collberg and Thomborson, 2002):

1. P contiene un *watermark* dinámico, y el código que construye dicho *watermark* ha sido alterado.

2. Un virus ha sido adjuntado a P.

3. P es una aplicación de e-*commercey* (comercio electrónico) la seguridad de su contenido se ha visto comprometida.

Para prevenir estas modificaciones indebidas, se puede añadir un código que realice tareas de *tamper-proofing* al programa. Este código debe detectar las modificaciones no autorizadas, y además debe provocar el fallo de la aplicación cuando las alteraciones sean evidentes. Idealmente, la detección de las modificaciones y el fallo de la aplicación deben estar dispersos en el tiempo para intentar confundir al atacante.

Hay tres modos básicos de detectar el *tampering*:

1. Examinando el propio ejecutable y compararlo con el ejecutable original para verificar que sean idénticos, se mide la velocidad a través de test. Se verifica si se utilizó algún algoritmo criptográfico como el Algoritmo de Resumen del Mensaje 5 o más conocido por sus siglas en inglés: MD5 (*Message-Digest Algorithm 5*).

2. Examinando la validez de los resultados intermedios producidos por el programa.

3. Generando el ejecutable rápidamente, con la esperanza que los mínimos cambios aplicados al programa generador produzcan un código que no se pueda ejecutar.

Marcas de agua para *software* (*software watermark*ing):

En el lenguaje coloquial es la vía para proteger un *software* a la falsificación de la propiedad intelectual por lo que se define al *software watermark*ing en empotrar un mensaje secreto dentro de otro mensaje. Esta técnica no impide realizar copias ilegales de un objeto, sino que nos permite probar la autoría de un contenido digital. Los autores Collberg y Thomborson definieron que en el *watermark*ing multimedia, el mensaje secreto es la información de *copyright*, y el mensaje que la contiene es una imagen, audio o vídeo (Collberg and Thomborson, 1998).

En la presente investigación se abordan diferentes medidas contra posibles ataques llegando a la conclusión que ninguna medida es infalible pero cuando se trata de proteger la propiedad intelectual del autor la opción más certera es la del *software watermark*ing.

1.2 *Software watermarking*

El *software watermarking* puede ser considerado como una variación del *watermarking* multimedia, que inició su navegación alrededor de 1954, aunque fue a partir de la década de los 90 cuando hizo un progreso considerable. Desde entonces se convirtió en una técnica muy popular para la protección de los derechos de autor del contenido multimedia. El *software watermarking* empezó desarrollarse a partir de los años 90. La patente de (Davidson and Myhrvol, 1996) presentó el primer algoritmo de *software watermarking* publicado, aunque los conceptos preliminares de *software watermarking* también se presentaron en (Moskowitz and Cooperman, 1994) y (Grover, 1997). Collberg y Thomborson presentaron definiciones detalladas para *software watermarking* en (Collberg and Thomborson, 1998) y (Collberg and Thomborson, 1999). En la actualidad las técnicas

de *software watermarking* han recibido menos atención que las técnicas aplicadas al contenido multimedia. En el *software watermarking* el mensaje que contiene la información de *copyright* es el código y los datos de un programa lo cual no debe afectar la funcionalidad del *software*.

El *software watermarking* se puede describir básicamente como empotrar una estructura W en un programa P de modo que W puede ser localizado y extraído de P de una forma fiable incluso después que P haya sido sometido a transformaciones.

Para que el *software watermarking* sea útil, éste debe ser resistente a la variedad de posibles ataques que puedan surgir. Se estima que un determinado atacante puede lograr su propósito dado a los suficientes recursos y el tiempo necesario. Por consiguiente, la meca de las técnicas de *software watermarking* es diseñar marcas que precisen de un tiempo, un esfuerzo y dinero considerable para ser destruidas. De este modo se intenta que resulte menos costoso adquirir una copia de forma legal, o escribir un programa propio, al de adquirirlo de forma ilegal y realizar despliegues del mismo.

1.2.1 Premisas del *software watermarking*

Las técnicas de *software watermarking* contienen tres premisas esenciales. La primera es la tasa de datos, que no es más que la mayor cantidad de datos ocultos posibles que pueden ser empotrados en el mensaje (programa). Es vital que la marca de agua codifique la mayor cantidad de información posible sin incrementar considerablemente el tamaño del código o del entorno de ejecución del programa. La segunda premisa es la imperceptibilidad, que de esta forma demuestra el grado de capacidad que tiene la información embebida para pasar desapercibida ante los ojos de un observador. Y por último la marca debe tener la resistencia frente a diferentes ataques de un adversario. En cualquier técnica existe un compromiso entre estas tres premisas, de modo que una tasa de datos alta implica baja imperceptibilidad y resistencia, o también incrementar la

resistencia aplicando redundancia en la información tiene como consecuencia una baja tasa de datos (Tomas, 2006).

1.2.2 Diferentes tipos de ataques a *software watermarking*

En general, no existe ninguna técnica de *watermarking* inmune a todo tipo de posibles ataques, y muchas veces deben emplearse varias técnicas simultáneamente para alcanzar el grado deseado de resistencia. Un escenario de los diferentes tipos de ataques, sería: Alice embebe una marca W con una clave K en un objeto O y después lo vende a Bob. Antes de que Bob pueda vender O a Douglas, Bob tiene que asegurarse de que ha conseguido inutilizar el *watermark*, ya que, de lo contrario, Alice será capaz de demostrar que sus derechos de la propiedad intelectual han sido violados.

Los cuatros principales tipos de ataques que se le pueden realizar a un *watermark* son:

• **Sustracción:** Si Bob puede detectar la presencia y la localización (aunque aproximada) de W, podría intentar extraerlo de O.

• **Deformación:** Si Bob no puede localizar W y está dispuesto a aceptar una pequeña degradación en la calidad de O, puede aplicar transformaciones que distorsionen uniformemente el objeto y por extensión cualquier *watermark* que pueda contener.

• **Adición:** Bob puede aumentar O insertando su propio *watermark* W’ de modo que Alice no puede demostrar que su marca precede temporalmente a la de Bob.

Alice puede ser capaz en algunos casos de realizar *tamper*-proof sobre su objeto contra posibles ataques de Bob. Hacer *tamper-proof* hace referencia a cualquier técnica usada por Alice específicamente para conseguir que los ataques contra su *watermark* sean inefectivos.

Confabulación: es cuando el ataque consiste en comparar varias copias de un programa, que sean de diferentes proveedores, lo cual permitirá al atacante estar en condiciones de detectar dónde está la marca.

1.2.3 Aspectos en el diseño de una técnica de *software watermarking*

El diseño de una técnica de *software watermarking* debe dar respuesta de forma general a las premisas esenciales del mismo por lo que se considera tres aspectos a tener en cuenta en el proceso de diseño (Tomas, 2006).

1. Tasa de datos: Qué tamaño tiene la marca en comparación con el tamaño del programa.

2. Forma del programa que lo contiene: En qué tipo de código será distribuido el programa: Código interpretado, para ejecutarse sobre una máquina virtual, o código binario nativo.

3. Modelo de amenazas esperado: Que tipos de ataques podemos esperar por parte de Bob para atacar el *watermark*s. Dependiendo de los recursos necesarios que pueda tener Bob para llevarlos a cabo.

1.2.4 Clasificación del *software watermarking*

Las técnicas de *software watermarking* pueden clasificarse en varios grupos:

Por su propósito: se dividen en *Watermark*s de prevención: previene el uso no autorizado del *software*; w*atermark*s de aserción: permiten hacer una reclamación publica de la propiedad del *software*; *watermark*s de permiso: permiten realizar copias o pequeñas modificaciones al *software*; *watermark*s de afirmación: aseguran la autenticidad del *software* que recibe el usuario final (Nagra, 2005).

Por el nivel de fragilidad: se dividen en robustas: la cual el *watermark* puede ser extraído incluso si se ha sometido a transformaciones, este tipo de *watermark*s es utilizado para prevenir usos no autorizados y para realizar una reclamación pública del mismo; los frágiles el *watermark*s siempre se destruirá cuando el *software* haya sido modificado, lo cual permite verificar la integridad del *software*, además se pone en práctica en *software* que permiten copias o pequeñas modificaciones al mismo (Collberg and Thomborson 2002).

Por su visibilidad: se dividen en dos grupos visibles o invisibles. En los *watermark*s visibles, existe una secuencia de entrada específica que hará que el código del *watermark* genere un mensaje visible al usuario, para poder demostrar la autoría del mismo (aserción), o para asegurar la autenticidad del *software* (afirmación). El *watermark*s invisibles no produce ningún tipo de señal visible, pero pueden ser extraídos usando algoritmos que no están directamente en manos del usuario final. Estos son usados en permisos y prevención (Zhu, 2007).

Por su técnica de extracción: se dividen en *watermark*s estáticos y dinámicos. El primero es insertado código del programa o área de datos y para su extracción no es necesario que el programa se ejecute y al contrario del primero este se almacena en el estado de ejecución de un programa, en lugar del propio código (Collberg and Thomborson, 1999).

El autor asume para su investigación la clasificación, según su técnica de extracción siendo por esta vía la forma de obtener el mensaje empotrado en un *software*.

1.3 Marcas de agua de *software* estático

Un método simple de *software watermarking* estático fue descrito 1994 por Holmes, este método consistía en la copia maestra del programa que contiene un segmento de datos que es no usado por el mismo. La localización y tamaño de este segmento sin usar son

determinados cuando el programa es conectado. Cuando una copia de este programa está hecha para distribución autorizada, este segmento es sobrescrito con la información del *watermark*s, como la fecha, la hora, y el destino de la copia. Holmes propone este método para la distribución en *Internet* de un programa.

En la actualidad los *watermark*s son guardados en el ejecutable de la aplicación, los cuales corresponden a la sección de datos inicializados (lugar donde guardan los *strings* estáticos), la sección de texto (código ejecutable) o la sección simbólica (información de *debug*) del ejecutable. Con la tendencia emigración a *software* libre y los entornos de desarrollo de alto nivel, se daría el caso de una aplicación *Java*, la información podría ser ocultada entre las varias secciones de una *class*.

Existen dos tipos principales de *watermark*s estáticos: *watermark*s de código, los cuales son almacenados en la sección del ejecutable que contiene instrucciones; y *watermark*s de datos, los cuales se almacenan en cualquier otra sección, incluyendo *headers*, secciones de *strings*, secciones de información de *debug* (Tomas, 2006).

***Watermark*s estáticos de datos**

Los *watermark*s estáticos de datos tienen una gran facilidad para ser construidos y reconocidos, ejemplo de lo antes mencionado es cuando la información del *copyright* puede estar contenida en una imagen JPEG lo cual permite fácilmente su extracción del código de un navegador (*netscape*) obteniendo todos los *strings* definidos y filtrando por los que contienen la palabra "*copyright*". Un segundo ejemplo es empotrar el watermark en una imagen (o audio o video digital) usando un algoritmo de *watermarking* de multimedia y luego almacenar la imagen en la sección de datos estáticos del programa. Este tipo de *watermark*s son muy vulnerables a ataques de deformación por ofuscación. Por lo que un ofuscador podría dividir todos los *strings* (y otros datos estáticos) en *substrings* que serían repartidos por el ejecutable; esto haría el reconocimiento del *watermark* casi imposible. Un método complejo sería convertir todos los datos estáticos en un programa que produjese los datos (Tomas, 2006).

*Watermark*s estáticos de código

Los *watermark*s de código se construyen de forma similar, dado que el código de objeto también contiene información redundante. Si no existen dependencias de datos o de control entre dos instrucciones de código consecutivas, éstas pueden ser dispuestas en cualquier orden. Entonces, un *bit* de *software watermarking* podría ser codificado dependiendo de si las dos instrucciones están en orden lexicográfico o no. Este método fue usado por International *Business Machines Corporation* (IBM) el cual usó una variación de esta técnica para construir una *watermark*s en su *software* consistente en el orden en que los registros eran puestos y sacados de la pila. Otro método propuesto consiste en codificarlo en el orden de las opciones de varios bloques. Davidson propuso un método innovador en el cual un número de serie se codificaría en la secuencia básica de bloques de los grafos de control de flujo del programa (Davidson, 1996).

Esto consistía en un método para generar y auditar firmas para módulos ejecutables en el que cada copia contiene una firma única que la distingue. La firma de cada copia autorizada, se codifica en el orden de las instrucciones del módulo ejecutable teniendo en cuenta que cada módulo ejecutable se compone de múltiples bloques de instrucciones. Para insertar la firma se selecciona un grupo de bloques que sigan un flujo de ejecución y se reordenan, posteriormente se modifican adecuadamente para no romper el flujo de ejecución, y por último se substituyen los bloques originales por los modificados. La copia modificada es funcionalmente equivalente a la original pero ahora los bloques reordenados proporcionan una firma única (Davidson and Myhrvold, 1996).

Este método fue una de las primeras propuestas formales de *software watermarking*. Su principal característica es su sencillez, pero al mismo tiempo su debilidad, el mismo es fácil de atacar mediante optimizaciones de código, mediante simples reordenaciones aleatorias de bloques básicos, o mediante ofuscaciones de código: insertar sentencias condicionales que siempre sean ciertas. En el caso de reordenaciones aleatorias de bloques básicos, se trata de un ataque por adición, es decir, si un atacante reordena de

nuevo los bloques básicos del módulo ejecutable, estaría introduciendo su propia firma y por tanto sería totalmente imposible recuperar la firma original (Tomas, 2006).

1.3.1 Tipos de algoritmos de *software watermarking* estático

Software watermarking a través de asignación de registros (*Watermarking through Register Allocation*)

En 1998 los autores Qu and Potkonjak crearon el algoritmo QP el mismo es una técnica de *software watermarking* que se basa para empotrar la información en la solución de coloreado de grafos. En compilación el coloreado de grafos se usa para asignar registros a las variables de un programa. Estos son grafos de interferencia que modelan la relación entre las variables que intervienen en un procedimiento, cada variable que interviene en el procedimiento se representa por un vértice, y si dos variables tienen rangos de vida solapados, entonces hay una arista que las une. Que dos variables tengan rangos de vida solapados significa que las dos variables coexisten en el tiempo, y por lo tanto no se le puede asignar el mismo registro porque se estaría eliminando a una de ellas.

La problemática del coloreado de grafos consiste en colorear el grafo de tal modo que se minimice el número de registros requeridos (número de colores) e impedir que dos variables compartan registro en el mismo instante (impedir que dos vértices unidos por una arista tengan el mismo color). El *watermark* está codificado en forma de colorear el grafo, o lo que es lo mismo, en cómo se asignan los registros a las distintas variables (Ginger, 2003).

Ginger crea el algoritmo QPS que surge como modificación al algoritmo QP, e intenta instaurar criterios de empotrado más estrictos. Se quiere evitar que un vértice pueda ser usado múltiples veces, la idea es seleccionar tripletas de vértices que formen unidades aisladas, de modo que no afecten a otros vértices del grafo. A estos conjuntos de tres vértices se les denomina tripletas coloreadas, por tanto, no existen aristas de unión entre ellos y tienen el mismo color. Este algoritmo puede ser aplicado a un programa entero o a un archivo *class.*

Watermarking en el dominio frecuencial

El *software watermarking* en el dominio frecuencial intenta aprovechar lo que en principio parece una limitación para crear una manera muy robusta de empotrar información utilizando el lenguaje de máquina. Este representa el código de maquina una sucesión lineal de código, sino como un objeto estadístico. Lo que en realidad vamos a marcar es la función que representa a las frecuencias de grupos de instrucciones que son ejecutadas. De esta manera se consigue esparcir el *watermark* por toda la aplicación, haciéndolo más imperceptible y en cierto modo más resistente frente a ciertos ataques.

El primer algoritmo desarrollado basado en esta técnica se denominó SHKQ y sus creadores fueron Stern, Hachez, Koeune y Quisquater en 1999, el cual proponía la idea de implementar el algoritmo en lenguaje ensamblador x86 (Stern, et al. 1999). A este mismo se le hizo una modificación para aplicarlo a *Java Bytecode* (Hachez, 2003), este algoritmo consiste en cambiar la posición de alguna variable local en la tabla, con lo que se consiguen cambiar instrucciones en el código.

A este último algoritmo se le hicieron modificaciones para integrarlo a la herramienta *Sandmark* la cual es una aplicación de Java. Los autores que propiciaron eso fueron (Tapas and Collberg, 2004), este algoritmo consiste en implementar el *watermark* usando un *codebook* de posibles modificaciones para un conjunto de grupos de hasta cuatro instrucciones *(opcodes*), llamados vector *groups*. Además, el *watermark* se codifica a lo largo de toda la aplicación.

Software watermarking vía predicados opacos

Los predicados opacos es una técnica estática de código. Su funcionamiento consiste, en empotrar un mensaje en el código no funcional, que solo sirve como recipiente del *watermark*. Es decir, dado un código a marcar, le añadimos porciones de código que son la propia marca. Para evitar la detección de la marca se utilizan predicados opacos, las cuales son construcciones lingüísticas que transforman saltos incondicionales en saltos que parecen, a ojos de un posible atacante, condicionales. Además, si se utilizan los

predicados opacos de manera adecuada, se podrán usar sus propiedades para camuflar otras estructuras, como los métodos de una clase en *Java*. En resumen, los predicados opacos son estructuras de código cuyo resultado es difícil de analizar.

Definición: Un predicado P es opaco en p si su resultado es conocido en tiempo de ofuscación. Denotamos P_p^F si P siempre evalúa Falso en p, P_p^T, si siempre evalúa Cierto, y $P_p^?$ si a veces evalúa Cierto y a veces Falso (Monden et al, 2000).

En 1998 surge una técnica llamada los métodos falsos (*dummy methods*) para codificar el *watermark*, basándose en los predicados opacos para hacer que los métodos no parecieran falsos (Monden et al, 2000).

La codificación por este método consiste en tres fases, primero se inyecta el método en el código fuente, luego se compila y finalmente se inyecta el *watermark* en el método ya compilado.

Static Graph-Based Watermarking

En este caso el *Graph Theoretic Watermarking* (GTW) utiliza el grafo de control de flujo de un programa (CFG): es la formación de los nodos y aristas, que representa el flujo de ejecución de un programa. Los nodos del CFG son los bloques básicos de ejecución del programa, y las aristas son los saltos entre bloques básicos de ejecución. A su vez, los bloques básicos de ejecución de un programa son conjuntos de instrucciones que se ejecutan secuencialmente, en los cuales la primera instrucción es el único acceso al bloque y la última instrucción es la única de salida (Venkatesan, 2001).

1.3.2 Diferencias entre los distintos tipos de algoritmos de *software watermarking* estático

Los algoritmos QP y QPS utilizan la asignación de registros como herramienta para codificar un mensaje, estos dos últimos surgen como mejora del algoritmo QP, lo que aumenta su grado de imperceptibilidad por lo que dificulta la ubicación del *watermark*, tienen una baja taso de datos, aunque tiene poca resistencia ante ataques simples.

En el caso del algoritmo SHKQ y sus dos variantes se puede concluir que la primera variante del mismo se basa en la modificación de los índices de la tabla de las variables locales de los métodos, haciendo variar los *opcodes* que incluyen dichos índices, al contrario de la segunda variante que requiere de un diccionario de modificaciones más complejo para sustituir o insertar grupos de hasta cuatro instrucciones (Tapas and Collberg 2004). Este último es más flexible a la hora de escoger elementos para el vector de frecuencia, ya que el primero se reduce a los *opcodes* que incluyen el operando de la tabla de variables locales.

El autor de la investigación considera que una gran desventaja que presenta el algoritmo que usa predicados opacos mediante el método de dummy falso, es que cuando se le realiza ataques automatizados: ofuscación, decompilación o recompilación, es posible sustraer la marca. Por otra parte, analizando el algoritmo GWT presenta con respectos a los anteriores un punto débil al no poder ser imperceptible. Lo cual lo hace vulnerable a los ataques manuales y por sustracción que generalmente son los de menor peligro al requerir la identificación del *watermark*. Al generar un alto porcentaje de operaciones aritméticas además de introducir código perceptible para implementar la adición de aristas.

1.4 *Software Watermark*s dinámicos

Los *watermark*s dinámicos, se almacenan en el estado de ejecución de un programa, en lugar del propio código. Esto hará que en muchos de ellos podamos aplicar *tamper-proofing* para neutralizar ataques por ofuscación. En los *watermark*s de *software* dinámico la aplicación O es ejecutada con una secuencia de entrada predeterminada que hace que el programa entre en un estado que representa el *watermark*. Los distintos métodos se diferencian según la parte del estado del programa en la que se almacenen, y según la manera de extraerlos (Collberg and Thomborson, 1999).

*Watermark*s de Huevos de Pascua

Este *watermark*s realiza una acción que es inmediatamente perceptible para el usuario, haciendo que la extracción de la marca sea trivial. El *watermark* consiste en una porción

de código que sólo se ejecuta si el usuario ejecuta la aplicación con una secuencia de entrada determinada (Collberg and Thomborson, 1999).

***Watermark*s de Estructuras Dinámicas de Datos**

Este tipo de *watermark* se ubica en tiempo de ejecución en la *heap* (memoria dónde se cargan los objetos), la pila (*stack*) o en los datos globales. Esto sólo ocurre cuando se ejecuta dicho programa con una secuencia de entrada determinada. La forma de extracción es examinando los valores que contienen las variables del programa una vez se ha alcanzado el final de los datos de entrada, esto se logra con una rutina específica que se ejecuta en el programa, o con el debuggado del mismo. El programa no produce ninguna salida de datos, no es inmediatamente perceptible para un atacante cuando se introduce la secuencia específica de entrada (Collberg and Thomborson, 1999).

***Watermark*s de Traza de Ejecución**

El *watermark* es empotrado en la traza de ejecución del programa (instrucciones, direcciones, o ambas) cuando éste se ejecuta con una secuencia de entrada determinada. El *watermark* se extrae mediante la monitorización de algunas propiedades estadísticas de la traza de direcciones de entradas/salidas de la secuencia de operadores ejecutada (Collberg and Thomborson 1999).

1.4.1 Tipos de algoritmos de *software watermarking* dinámico

Dynamic Graph-Based Software watermarking

Este algoritmo fue desarrollado por los autores Collberg y Thomborson al cual denominaron CT. El mismo se encuentra implementado en *Java* e incluido en la herramienta *SandMark*. Dicho algoritmo utiliza estructuras dinámicas de datos para codificar la marca, esto consiste en incrustar la marca en la topología de un grafo construido de forma dinámica en memoria, y en tiempo de ejecución, el cual se divide en subgrafos. En la extracción de la marca, se asume una clave secreta K, que es la secuencia de entrada de la aplicación (Collberg, 2004).

Un diseño basado en grafos dinámicos trae grandes ventajas:

- El parecido entre el código de la aplicación y del *watermark.*
- La posibilidad de empotrar *watermark*s de gran tamaño.
- Las posibilidades de aplicar técnicas de *tamper-proofing.*

Otra gran ventaja de este diseño viene dada por la posibilidad de dividir un grafo muy grande en varios subgrafos más pequeños. Lo cual permite esparcir el *watermark* a lo largo de toda la aplicación, lo que posibilita el gran tamaño del *watermark* sin que disminuya el grado de imperceptibilidad del mismo. El *watermark* se encuentra codificada en forma de datos, el código que generado, es el resultado de su ejecución.

El resultado de la ejecución de este código genera una estructura de datos que se almacena en memoria, por lo que, si se diseña apropiadamente estas estructuras, se podrán dotar de propiedades específicas para luego insertar código que verifique las propiedades de las mismas.

Threading Software watermarking

Esta técnica se basa en introducir nuevos *threads* en secciones del programa en las que sólo se ejecuta un único *thread*. En un programa con múltiples *threads* ejecutándose simultáneamente, se puede dar el caso en el que varios de ellos intenten leer o escribir sobre datos compartidos, o de forma más general, intenten usar recursos compartidos simultáneamente. Cuando se da esta situación, el resultado final de la ejecución dependerá únicamente de cómo han sido programados los distintos *threads*. Lo cual permite que múltiples *threads* compartan recursos de manera controlada (Nagra, 2007).

Este algoritmo consta de tres fases y está realizado sobre *Java Bytecode*

1. *Tracing*: Se analiza y captura la traza de ejecución del programa dada la entrada secreta I.
2. *Embedding*: Se selecciona un número W (el *watermark*) y se empotra en la secuencia de ejecución de los distintos *threads*.

3. *Recognition*: Se ejecuta el código con la entrada secreta I, se analiza la traza de ejecución y se extrae el *watermark* W.

1.4.2 Diferencias entre los distintos tipos de algoritmos de *software watermarking* dinámico

El autor de la investigación, analizando los dos algoritmos antes expuestos, concluye que el algoritmo CT consta de pocas prestaciones al no dividirse el grafo en varios subgrafos que se distribuyan a lo largo del camino (*path*) de ejecución dada la entrada secreta. Ya que las operaciones necesarias para construir el grafo no resultan sospechosas por sí solas, un gran número de ellas concentradas en un mismo punto pueden llegar a serlo. El algoritmo CT está diseñado para resistir ataques automatizados (optimizaciones, ofuscaciones), que el marcado es demasiado largo para realizar ataques por inspección manual.

El algoritmo *Threading Software watermarking* muestra que el *watermark*s es resistente especialmente a ataques por ofuscación, y ataques por decompilación y recompilación. En cambio, los ataques por adición representan la amenaza más eficaz para este tipo de *watermark*.

1.5 Diferencias entre la estenografía, criptografía y el *software watermarking*

La estenografía es un área similar a la de criptografía la cual proviene del griego *stegos* (ocultar) y *graphos* (escritura) en la que es un conjunto de técnicas que permiten ocultar o camuflar cualquier tipo de datos, dentro de información considerada como válida (Gómez, 2009).

Gerardino en el 2007 define como estenografía a la técnica y el arte de ocultar una información dentro de otra, demostrando que solo quienes conozcan cierta información de esa ocultación estará en condiciones de descubrirla.

Hardikkumar (2012) define como criptografía al conjunto de técnicas con el objetivo básico de alterar símbolos de la información sin transformar su contenido. Por su parte Goel (2008) plantea refiriéndose a la criptografía que esta trata de proteger el contenido de los mensajes. Así mismo, (López, 2008) considera que la criptografía cifra la información para que no sea comprensible por las entidades no autorizadas, incluso aunque conozcan la existencia de esa información, es decir es el estudio de métodos para enviar mensajes en secreto (cifrados) para que sólo el destinatario al cual va dirigida la información pueda quitar el cifrado y pueda leer el mensaje.

Haciendo un análisis de la diferencia entre la criptografía y la estenografía asumiendo los conceptos mencionados por Gómez (2009), Goel (2008) y López (2008), se arriba a la conclusión de que en la criptografía se modifica el mensaje original, de forma que no resulte entendible a quien no conozca la forma de descifrarlo, pero no hay ningún interés en ocultar la existencia y el acceso al mensaje cifrado, aunque éste resulte ilegible. No siendo esto así en la estenografía la cual intenta ocultar la existencia misma del mensaje.

Aun cuando las marcas de agua para *software* pertenecen a la estenografía, entre ellas presentan sus diferencias (Areitio, 2004).

Gómez (2009) establece las siguientes diferencias entre el *software watermarking* y estenografía. El *software watermarking* tiene como objetivo la protección de la propiedad intelectual, no siendo así en la estenografía, y la estenografía a su vez no presenta robustez contra un borrado hostil, o destrucción.

Gerardino en el 2007, estableció las siguientes diferencias entre la estenografía y el *software watermarking* en lo cual expresó que, dependiendo de su aplicación y la intencionalidad de los mismos, la estenografía está relacionada con el mundo de los comportamientos ilícitos o de espionaje. Siendo así que las marcas de agua de *software* obedecen fundamentalmente a intereses de mercado.

El autor de la investigación coincide con lo expresado por Gómez, ya que en el caso de Gerardino no coincide con las últimas dos diferencias puesto que un comportamiento ilícito puede estar dado por la violación del derecho de autor.

1.6 Compartición de secretos

Los protocolos de compartición de secretos fueron creados de manera independiente por Blakley. Su principal objetivo es que exista una llave secreta que provee acceso a muchos archivos de actividades muy importantes (Blakley, 1979).

Dicho protocolo surge de la necesidad del manejo de llaves robustas. Dada una llave criptográfica se quiere generalizar la noción de control de acceso fragmentado. Una problemática es si la llave de acceso se pierde, o bien se borra o destruye la memoria de la computadora que almacena la llave, entonces los archivos importantes se vuelven inaccesibles. Este protocolo permite fragmentar mucho la llave y distribuirla a varios usuarios lo cual en caso de extraviarse en algunos usuarios con al menos un numero de ellos que la conserven es posible volver a obtener los datos cifrados, permitiendo la confiabilidad sin riesgo creciente (Vázquez, 2004).

1.7 Conceptos Matemáticos. Los grafos y la interpolación polinómica

Conceptos de grafos

Un grafo G es un par $G = (V, E)$, donde V es un conjunto finito (vértices, nodos) y E es un multiconjunto de pares no ordenados de vértices $\{x, y\}$, que se denominan arcos, aristas, lados. Entonces x y y son extremos de $\{x, y\}$. Denotamos $V\ (G)$ por el conjunto de vértices del grafo G y por $E(G)$ el conjunto de aristas del grafo G. Además, $\nu\ (G)\ y\ \varepsilon(G)$ denotan el número de vértices y el número de aristas de G respectivamente. E es un multiconjunto y es posible que existen pares repetidos, en este caso G tiene lados múltiples. También es posible que algún par no ordenado de E tenga el mismo vértice repetido, en este caso decimos que es un lazo (*loop*) o bucle. Cuando existen múltiples lazos decimos que G es un multígrafo. Si no hay aristas múltiples ni lazos decimos que es un grafo simple (Claverol et al., 1999).

Grafo simple

Un grafo simple G es una estructura que consta de un par ordenado de conjuntos $(V; E)$, siendo V distinto de 0. Los elementos de V se llaman vértices y los elementos de E se llaman aristas. En un grafo simple, una arista es un par $\{x, y\}$ no ordenado de vértices diferentes (Claverol et al., 1999).

Árboles

Los grafos tienen un tipo de grafo que los hace peculiares que son los denominados árboles, que se presenta en múltiples aplicaciones. En la computación los árboles son particularmente útiles ya que se utilizan para organizar información de tal modo que sea posible efectuar eficientemente operaciones que atañan a esa información. En la construcción de algoritmos eficientes para localizar artículos en una lista, a la hora de construir códigos eficientes para almacenar y transmitir datos, y cuando se modelan procedimientos que son llevados a cabo al utilizar una secuencia de decisiones (Claverol et al., 1999).

Interpolación polinómica

Mathews define que interpolar significa estimar el valor desconocido de una función en un punto, tomando una media ponderada de sus valores conocidos en puntos cercanos al dado (Mathews, 2000).

La interpolación tiene como objetivo la obtención de nuevos puntos a partir del conocimiento de un conjunto discreto de ellos (Fuentes, 2014). La interpolación polinómica es un método usado para conocer, de un modo aproximado, los valores que toma cierta función de la cual sólo se conoce su imagen en un número finito de abscisas. Algunas veces no se conocerá la expresión de la función y sólo se dispondrá de los valores que toma para dichas abscisas.

Existen varios métodos de interpolación polinómica como el de Newton: el cual permite realizar la interpolación en un punto de forma sucesiva: partiendo de dos nodos ir agregando los demás en el orden que se desee, este método permite sin realizar ninguna operación adicional ir obteniendo en cada paso del proceso una estimación del error de interpolación. La presente investigación se centra en el método de Lagrange:

Método de Langrage: brinda un algoritmo eficiente para hallar el polinomio interpolador, además de permitir encontrar el valor interpolado para una x especifica sin necesidad de hallar la expresión analítica del polinomio interpolador.

Dados un entero n no negativo, $n + 1$ puntos $x_0 \ldots x_n \in R$ distintos dos a dos y los correspondientes valores $f(x_0), \ldots, f(x_n)$ de una función, encontrar un polinomio $P(x)$ de grado $\leq n$ tal que

$$P(x0) = f(x_0);\ P(x_1) = f(x_1) \ldots, P(x_n) = f(x_n).$$

El problema de interpolación de Lagrange tiene solución única, que se denomina polinomio interpolador de Lagrange de grado $\leq n$ de la función f en los puntos $x_0 \ldots x_n, n + 1$ puntos $x_0 \ldots x_n \in R$, distintos dos a dos, y los valores $f(x_0), \ldots, f(x_n)$.

$$P(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} + a_n x^n$$

Coeficientes indeterminados

$$\begin{cases} a_0 + a_1 x_0 + \cdots + a_{n-1}x_0^{n-1} + a_n x_0^n = f(x_0) \\ a_0 + a_1 x_1 + \cdots + a_{n-1}x_1^{n-1} + a_n x_1^n = f(x_1) \\ \vdots \\ a_0 + a_1 x_n + \cdots + a_{n-1}x_n^{n-1} + a_n x_n^n = f(x_n) \end{cases}$$

Polinomios base de Lagrange, $Lj(x), j = 0 \ldots n$

$$Lj(x) = \frac{(x - x_0) * \dots * (x - x_{j-1}) * (x - x_{j+1}) * \dots * (x - x_n)}{(x_j - x_0) * \dots * (x_j - x_{j-1}) * (x_j - x_{j+1}) * \dots * (x_j - x_n)} = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i}$$

Expresión explicita del polinomio interpolador

$$P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x) = \sum_{j=0}^{n} f(x_j) \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i}$$

CAPITULO 2. DIAGNÓSTICO DE LAS MARCAS DE AGUA PARA *SOFTWARE* EN LA CARRERA DE INGENIERÍA INFORMÁTICA EN LA UNIVERSIDAD AGRARIA DE LA HABANA Y PROPUESTA DEL ALGORITMO

En este capítulo se muestra una breve caracterización de la carrera de Ingeniería Informática de la Universidad Agraria de La Habana “Fructuoso Rodríguez Pérez”, se presentan, además, los resultados del análisis documental realizado a las tesis de grado de los estudiantes sobre el uso de algoritmos de protección de *software watermarking* (marcas de agua para *software*), así como los resultados de las encuestas a profesores y estudiantes. En el epígrafe 2.1 se explica las 4 etapas las cuales está conformada el capítulo 2, de cómo se concibió la investigación. Seguidamente en el 2.2 se caracteriza la Carrera de Ingeniería Informática de la Universidad Agraria de La Habana, en la que se esgrimen los datos sobre la cantidad de estudiantes por año, además de la cantidad de tesis presentadas por año. En el 2.3 se plasman los resultados del diagnóstico de las marcas de agua para *software* aplicado en dicha carrera, la cual arrojó resultados de suma importancia para la investigación. En el 2.4 se realiza la propuesta del algoritmo de marca de agua para *software* basándose en el esquema de Shamir el cual brinda robustez frente a los diferentes tipos de ataques que se les puede realizar al *software watermarking*. En el 2.5 se expone el diseño del software watermarking propuesto y como debe ser insertado y extraído en un *software* y por último En el 2.6 se implementa el algoritmo de marca de agua para *software* propuesto.

2.1. Concepción de la investigación

La investigación llevada a cabo está conformada en cuatro etapas con el fin de desarrollar un algoritmo de marca de agua para *software* basándose en la revisión bibliográfica antes expuesta. Las etapas contempladas permitieron encausar la investigación siguiendo un orden lógico que permitió finalmente arribar al resultado esperado lo que permitirá contar con un algoritmo de *software watermarking* frente a la piratería con fines de lucro comercial.

Primera etapa: Revisión de los fundamentos teóricos y metodológicos referentes a las marcas de agua para *software* descritos por la comunidad científica nacional e internacional.

Segunda etapa: Revisión de las Tesis de Diploma en la carrera de Ingeniería Informática de la Universidad Agraria de La Habana del 2010 al 2015, para conocer en el caso de que el resultado final de las investigaciones fuera un *software* si se le incorporaba alguna técnica de marca de agua para *software*.

Tercera etapa: Aplicación de encuestas a estudiantes y profesores de la carrera de Ingeniería Informática de la Universidad Agraria de La Habana para medir el conocimiento sobre la temática abordada.

Cuarta etapa: Desarrollo de un algoritmo de marca de agua para *software* basándose en la interpolación polinómica específicamente Lagrange mediante el método de Shamir.

2.2. Caracterización de la Carrera de Ingeniería Informática de la Universidad Agraria de La Habana "Fructuoso Rodríguez Pérez"

En 1976 se crea la especialidad de Ingeniero en Sistemas Automatizados de Dirección Técnico Económico (SAD-TE). El Plan de Estudio A estaba diseñado de forma tal que este especialista se dedicara a la automatización de los procesos en empresas y dentro de esta se inclinaba hacia los procesos industriales, con el enfoque integral que definían los llamados Sistemas Automatizados de Dirección.

El Plan B significó un avance respecto al A, perfilándose mejor los ciclos de asignaturas, incluyéndose prácticas en microcomputadoras a partir de 1985, lo que permitió ampliar la formación y habilidades de los egresados de la especialidad.

El Plan C comenzó a aplicarse en el curso 90-91, creció también apreciablemente la cantidad de tiempo que debía dedicar el estudiante al trabajo con computadoras personales en la ejecución de los proyectos y trabajos extra clases.

Las sucesivas modificaciones del plan de estudio de la Carrera de Informática, sumado a aspectos propios como el cambio en los niveles de abstracción al manejar recursos y herramientas informáticas, la diversidad de enfoques locales en cuanto a tecnologías, lenguajes de programación y áreas de aplicación, el surgimiento de la carrera en todas las provincias, y la experiencia de las universidades que tienen dicha carrera en Cuba, en la realización de ajustes a los planes de estudio, a grupos de estudiantes relacionados con colectivos de desarrollo o investigación, acercan el plan actual a los requerimientos del Plan D, en cuanto a los enfoques de esencialidad, presencialidad y flexibilidad exigidos.

Este profesional se inserta de manera multidisciplinaria con especialistas de diversas ramas para concebir y desarrollar la solución informática que brinde respuesta a las necesidades del problema en cuestión, siendo capaz de asimilar los modelos correspondientes, seleccionar y utilizar el equipamiento, técnicas y métodos más efectivos para el procesamiento de la información.

Tiene su campo de acción asociado a la concepción, modelación, diseño, desarrollo, implantación, integración, mantenimiento y prueba de sistemas informáticos, explotando las infraestructuras de almacenamiento, procesamiento e intercambio de información disponibles, que contribuya al incremento de la eficacia y eficiencia en el funcionamiento de un amplio espectro de organizaciones, aplicando medidas organizativas y funcionales que propicien dicho objetivo, cumpliendo los estándares de calidad establecidos, prevaleciendo en todo lo anterior criterios que sustentan los altos intereses del país en la producción y los servicios. Los egresados deben ser capaces de asesorar en la compra de hardware, programar lo especificado por otro, especificar para que otro programe, y dirigir programadores.

Los profesores del departamento de informática que contribuyen a la formación de los estudiantes de la carrera de ingeniería informática está formada por 25 trabajadores que incluye 1 secretaria, el resto son profesores.

En cuanto a las categorías docentes el departamento cuenta con 7 Profesores Asistentes, 16 Profesores Instructores y 2 adiestrados, según la categoría científica el claustro (sin contar adiestrados) cuenta con 1 Doctor en Ciencias y 6 Másteres en Ciencias.

En el curso 2014-2015 se cuenta con la siguiente matrícula:

Tabla 2.1 Matrícula de la carrera de Ingeniería Informática por años

Años	Curso regular diurno (CRD)	Curso por encuentro (CPE)
Primero	36	10
Segundo	14	20
Tercero	25	24
Cuarto	24	-
Quinto	10	13
Sexto		13
Total:	**118**	**81**

Fuente: Tomado de la Secretaría Docente de la Facultad

2.3. Diagnóstico de las marcas de agua para *software* en las tesis de la carrera de Ingeniería Informática

El diagnóstico realizado estuvo dirigido al análisis de las tesis de Diploma presentadas por los estudiantes de curso regular diurno y por encuentro de la carrera de informática de la Universidad Agraria de La Habana en un periodo del 2010 hasta el 2015, cuyo objetivo fundamental fue analizar si en las investigaciones concluidas cuyo resultado final fuera un *software*, a este se le incluía algún algoritmo de marca de agua para *software* como medida de protección basándose en la propiedad intelectual.

Tabla 2.2 Tesis presentadas por curso académico

Curso académico	Curso Regular Diurno	Curso por Encuentro	Sub-Total
2010-1011	11	2	13
2011-2012	11	18	29
2012-2013	7	3	10
2013-2014	7	11	18
2014-2015	8	7	15
Total	**44**	**41**	**85**

Fuente: Elaboración propia

Como muestra la tabla 2.2 en el curso regular diurno en el período que se realiza el diagnostico se presentaron 44 tesis, que representan el 52% del total, y en el curso por encuentro en mismo ciclo se realizaron 41 que representa el 48%.

Tabla 2.3 Resultados finales de las investigaciones llevadas a cabo en las tesis

Curso académico	*software*	Aplicaciones Web	Sub-Total
2010-2011	6	7	13
2011-2012	12	17	29
2012-2013	7	3	10
2013-2014	6	12	18
2014-2015	8	7	15
Total	**39**	**46**	**85**

Fuente: Elaboración propia

Después del análisis de un total de 85 tesis se pudo determinar que de ellas dieron como resultado final 39 *software* representando el 46% y 46 investigaciones proponen una aplicación web, significando el 54% detallado en la tabla 2.3. El análisis documental

realizado a las tesis cuyo resultado es un *software* evidencia que no se tuvo en cuenta las marcas de agua para *software* por lo que estos productos informáticos son vulnerables a una violación del derecho de autoría. Coincidiendo con (Collberg and Thomborson, 1998) es necesario el conocimiento de *software watermarking* como vía para proteger un *software* al que se le puede falsificar el derecho de autor.

2.3.1. Resultados de la encuesta aplicada a profesores y estudiantes de la carrera de Ingeniería Informática en la Universidad Agraria de La Habana

Para el diagnóstico se confecciono una encuesta (Anexo 1 y 2) que tuvo como propósito medir el grado de conocimiento en profesores y estudiantes acerca del dominio de las marcas de agua para *software*. Para este fin se definió como población un total de 24 profesores y 21 estudiantes. Se seleccionó una muestra no probabilística intencional, (Castellanos, 1998) constituida por 23 profesores que representan el 96% de los encuestados y 21 estudiantes que representa el 100% de la población.

Resultados de la encuesta a profesores

Los resultados de la encuesta a profesores permitieron conocer el dominio que poseen relacionado con los algoritmos de marca de agua de *software*, para ellos se realizaron diferentes preguntas.

En la figura 2.1 relacionado al conocimiento de ley cubana que proteja la propiedad intelectual de un *software*, se representa de la siguiente manera:

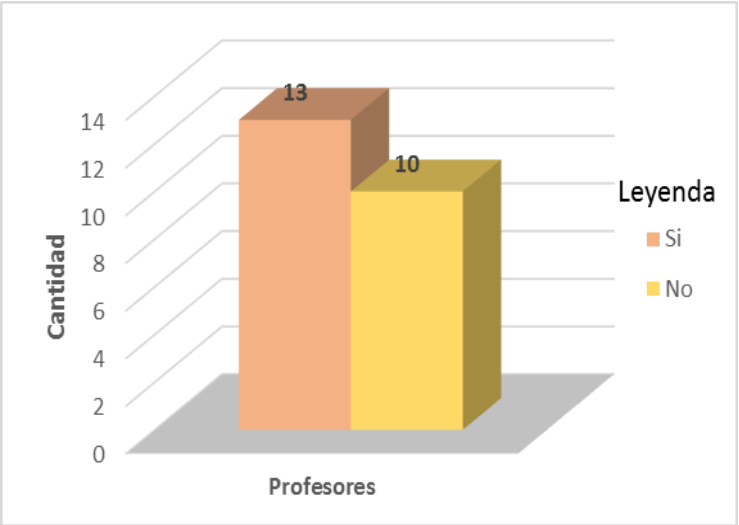

Figura 2.1. Conocimiento de ley cubana que proteja la propiedad intelectual de un *software*.

Fuente: Elaboración propia

Se puede apreciar que el 57% de los encuestados los cuales están representados por 13 docentes afirmaron tener conocimiento sobre el tema no así 10 docentes para un 43%. Lo que evidencia que en sentido general hay un dominio del reconocimiento de un derecho particular en favor de un autor u otros titulares, sobre los derechos de propiedad intelectual que protegen los derechos de los autores al garantizar sus investigaciones, teniendo en cuenta la necesidad de fomentar una protección eficaz, adecuada y de velar por las medidas y procedimientos destinados a hacer respetar los mismos, coincidiendo con lo dicho por la Organización Mundial de la Propiedad Intelectual (OMPI, 2015).

En la figura 2.2 se aborda el dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*) detallándose de la siguiente forma:

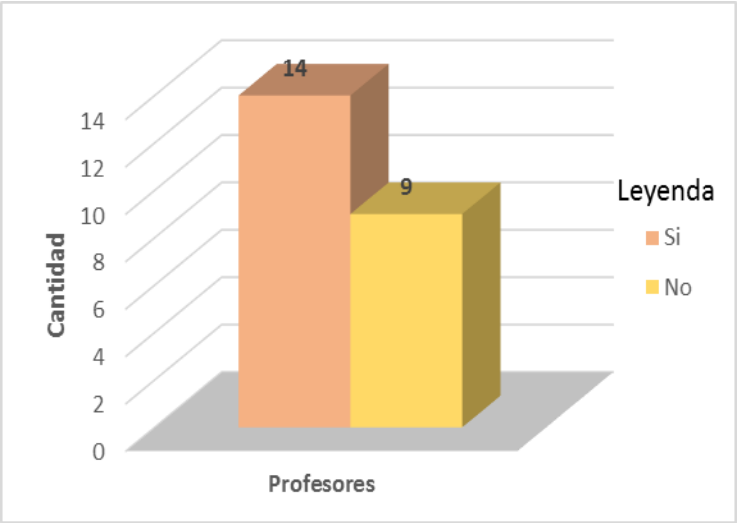

Figura 2.2 Dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*)

Fuente: Elaboración propia

En la misma se muestra que 14 docentes respondieron que si tienen dominio sobre el tema para un 60%, y 9 docentes plantean no tener dominio del mismo para una representación del 40%, presentan desconocimiento sobre los ataques de *software* que se le puede realizar a la propiedad intelectual contenida en los mismos. No viendo que, de esa forma, se ven comprometidos los atributos de privacidad e integridad de los componentes de *software* sometidos a un ataque. Coincidiendo con (Nuñez, 2013) cuando dice que los ataques al *software* crean manipulaciones y parches que alteran la integridad y la privacidad de los mismos, generando licencias no válidas y las copias de propiedad intelectual que afecta al atributo de la privacidad de la componente.

En la figura 2.3 se hace referencia al dominio de las marcas de agua para *software* (*software watermarking*) mostrándose a continuación:

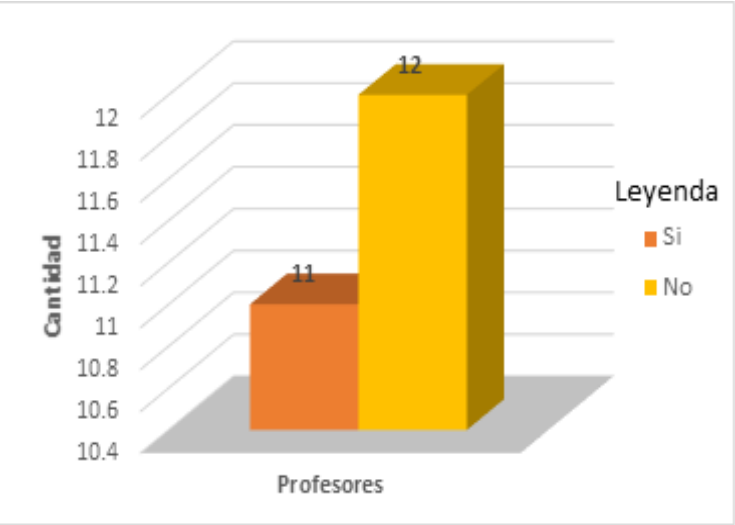

Figura 2.3 Dominio sobre el *software watermarking*

Fuente: Elaboración propia

Como se aprecia de los encuestados, 11 afirmaron tener dominio para un 47%, mientras que los 12 restantes docentes demostraron no tener dominio sobre dicho tema para una representación del 53% lo que demuestra la importancia del tema investigado, lo que demuestra que el *software watermarking* es una técnica que, aunque no protege ante copias ilegales, proporciona un método que permite demostrar la propiedad intelectual del software. Coincidiendo con lo que plantea Zhu en el 2007 cuando dice que el *software watermarking* trata de disuadir la piratería, adjuntando una marca de copyright en cada copia del *software* distribuido.

La figura 2.4 reseña el conocimiento acerca de los tipos de ataques que se les puede realizar al *software watermarking*.

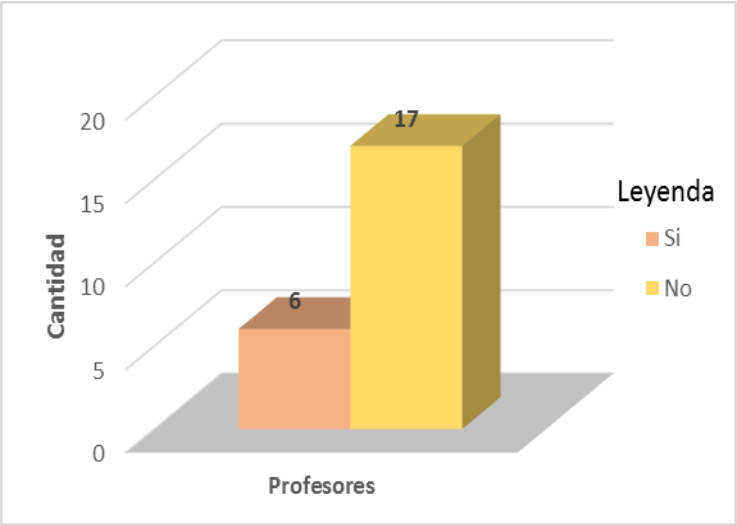

Figura 2.4. Conocimiento sobre los tipos de ataques al *software watermarking*

Fuente: Elaboración propia

En la misma se obtuvo que el 26% de los encuestados tienen conocimiento respecto al tema que corresponde a 6 docentes, no así los 17 docentes restantes para un 74%. Donde se evidencia el desconocimiento de los ataques de *software watermarking*: adición, sustracción, deformación y confabulación, aunque no existe ninguna técnica de *watermarking* completamente inmune a cualquier tipo de ataque, es necesario delimitar a que tipos de ataques nos vamos a enfrentar. Aquellos ataques que resulten efectivos a la hora de anular un *watermark*, solo se tendrán en cuenta si son factibles en términos de recursos necesarios.

La figura 2.5 se refiere a si conoce que en Cuba se ponen en funcionamiento algún algoritmo de *software* watermarking como contramedida frente a la piratería de *software*

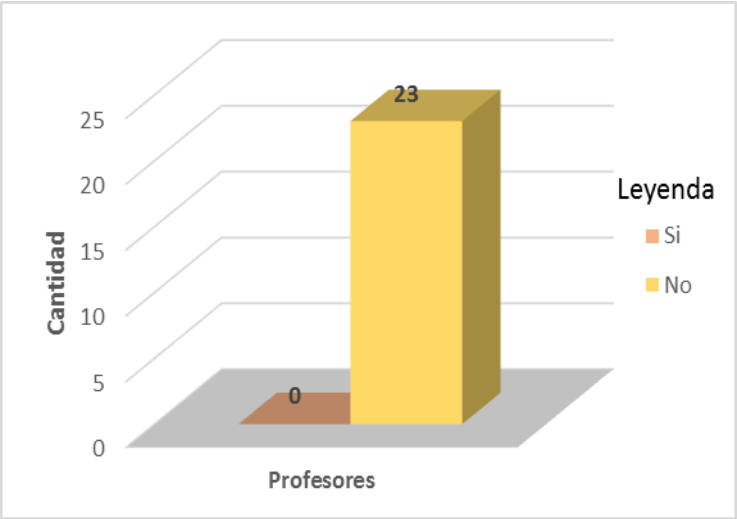

Figura 2.5. Conocimiento si Cuba pone en funcionamiento algún algoritmo de *software watermarking* como contramedida a la piratería de *software*.

Fuente: Elaboración propia

En ella se demostró que el 100% de los encuestados argumentó no tener conocimientos, representando a los 23 docentes. Esto implica que el desconocimiento de la misma influye en la protección de los derechos de autor en los *software* producidos en la carrera de informática de la UNAH. Demostrando lo que plantea (Orúe, 2002) cuando expresa que en los últimos años se ha incrementado el interés de la comunidad científica, por el establecimiento de las definiciones preliminares de los requisitos de un sistema de marcas de agua para *software* eficaz, con vistas a su estandarización. Sin embargo, queda mucho por hacer antes de que pueda hablarse de la adopción definitiva de un estándar para algunas aplicaciones y en el contexto local solo se ha hecho una revisión general de los conceptos básicos de las marcas de agua para *software*.

La figura 2.6 hace referencia a si en la Universidad se pone en funcionamiento algún algoritmo de *software watermarking* como medida frente a la piratería de *software* y en caso de utilizar alguno nombrarlos.

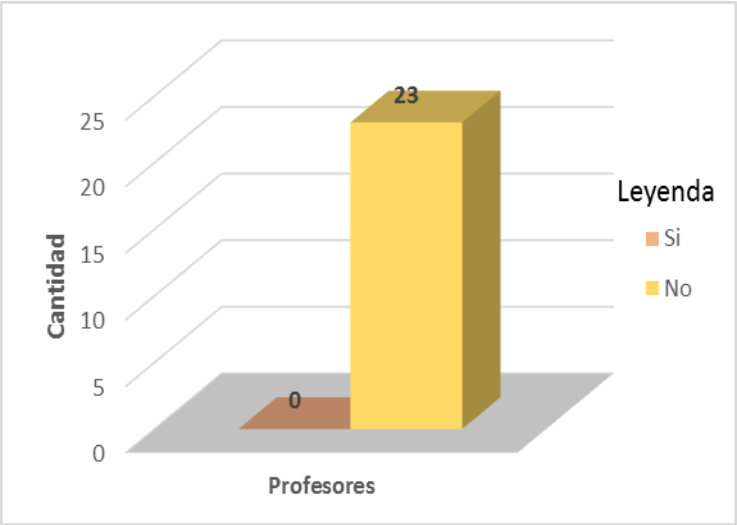

Figura 2.6. Conocimiento si en la universidad se utiliza algún de *software watermarking* como contramedida a la piratería de *software*

Fuente: Elaboración propia

De los 23 docentes respondieron en su totalidad para un 100% de que no conocían ningún algoritmo para este fin, teniendo correspondencia con la pregunta anterior.

A la pregunta referida al conocimiento de los algoritmos de marcas de agua para *software*QP, QPS, QPI, SHQK, SHQK2, CT, *Threading software watermarking*, 23 docentes encuestados no respondieron, representando 100% de los mismos, esto evidencia el desconocimiento que existe en el claustro de profesores sobre dicha temática. Se coincide con lo expresado por la Alianza de Negocios de *Software* (2013) que el desconocimiento de las leyes sobre la propiedad intelectual puede conllevar a una violación del *copyright* y por tanto pérdidas económicas al país en el que se produzca el delito.

A la interrogante referida a la utilización del algoritmo de marca de agua para *software* y mencionar en que producto fue aplicado, los 23 docentes encuestados no respondieron la interrogante para un 100%, poniendo de manifiesto el desconocimiento de esta temática para la protección del derecho de autor en los *softwares*, concordando con

(Zhu, Thomborson and Wang, 2005) cuando afirman que el uso de un algoritmo de marca de agua para *software* permite identificar la autoría del *software* cuando se hace una violación del *copyright* permitiendo la piratería del mismo.

Resultados de la encuesta a estudiantes

La figura 2.7 relacionada al conocimiento de ley cubana que proteja la propiedad intelectual de un *software*, se representa de la siguiente manera:

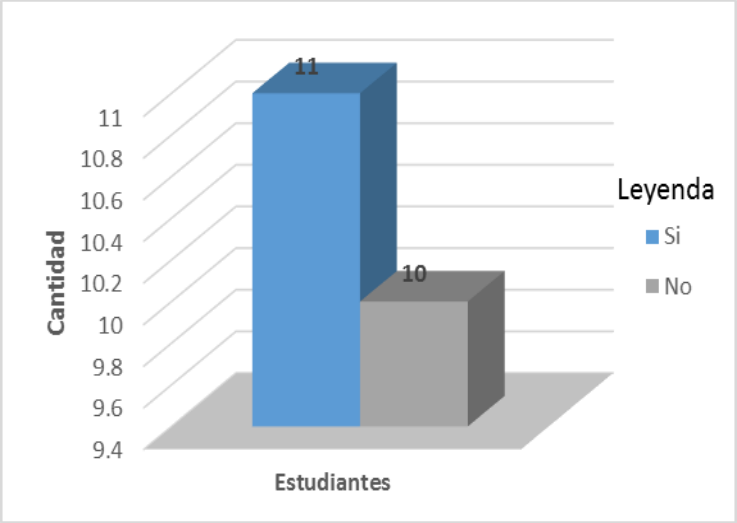

Figura 2.7 Conocimiento de ley cubana que proteja la propiedad intelectual de un *software*

Fuente: Elaboración propia

En la misma se aprecia que el 52% de los encuestados representados por 11 estudiantes afirmaron tener conocimiento sobre el tema no así el 48% para un total de 10 estudiantes. Se evidencia en la "gestión del conocimiento", que la prosperidad económica de los países se desarrolla sobre la base de los bienes intangibles, donde la universidad juega un papel esencial. Al potenciar los descubrimientos y los conocimientos teóricos básicos que son transmitidos a los estudiantes. Sumando de forma su potencial innovador y a su vez permitiendo tener herramientas que permitan la protección de los mismos (Hernández, 2013).

En la figura 2.8 representa el conocimiento relacionado al dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*).

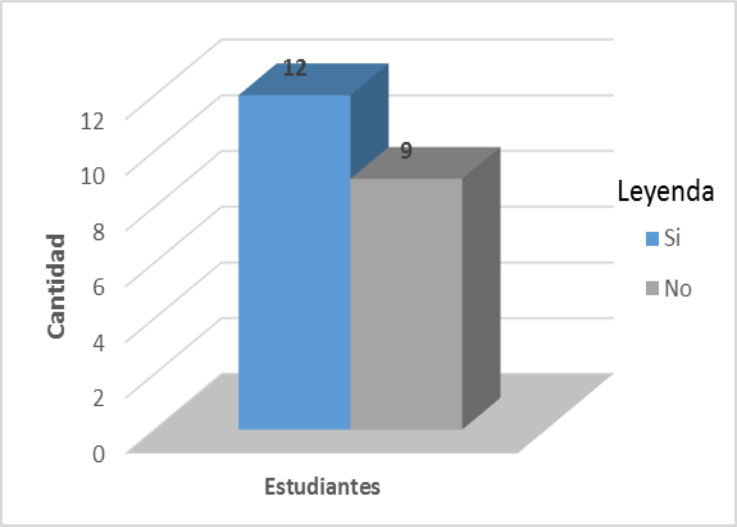

Figura 2.8 Dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*)

Fuente: Elaboración propia

Se representa a 12 estudiantes encuestados los cuales respondieron que, si tiene dominio sobre el tema para un 57%, y 9 estudiantes esgrimen no tener dominio del mismo para una representación del 43%. Demostrando la concordancia con la encuesta aplicada a docentes, en la que se demostró tener conocimiento respecto al tema, influyendo en los estudiantes de manera positiva en la formación y adquisición de los conocimientos para los futuros egresados de la carrera de ingeniería informática de la UNAH, obteniendo de esta forma un profesional competente.

En la figura 2.9 se hace referencia al dominio de las marcas de agua para *software* (*software watermarking*).

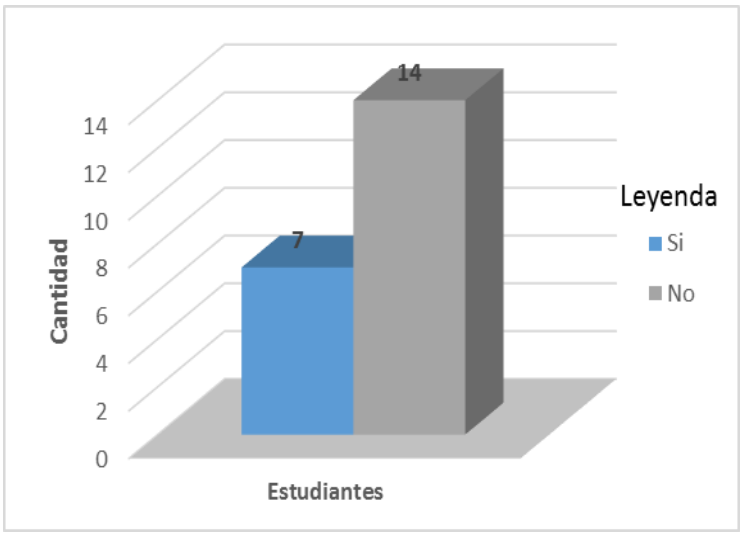

Figura 2.9 Dominio sobre el *software watermarking*

Fuente: Elaboración propia

Como se aprecia de los encuestados, 7 estudiantes afirmaron tener dominio para un 33% mientras que los 14 restantes estudiantes demostraron no tener dominio sobre dicho tema para una representación del 67%. Manteniendo relación con la encuesta aplicada a docentes en la que se evidencia falta de conocimiento sobre el tema e influye en la formación del estudiante el cual no adquiere los conocimientos necesarios para lo protección de un *software* basado en el derecho de autor.

La figura 2.10 está dirigida al conocimiento acerca de los tipos de ataques que se les puede realizar al *software watermarking*.

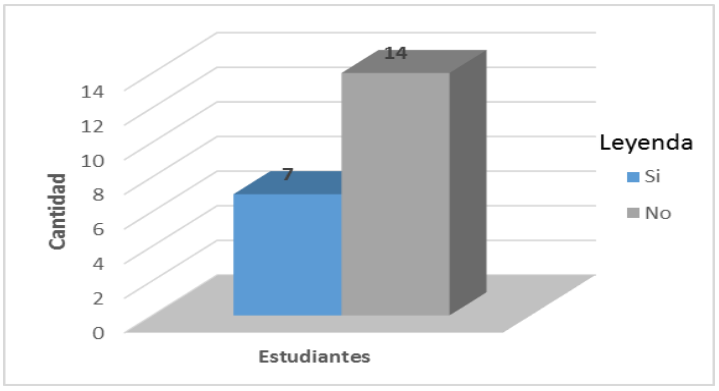

Figura 2.10 Conocimiento sobre los tipos de ataques al *software watermarking*

Fuente: Elaboración propia

La cual evidencia que, de los estudiantes encuestados, 7 de ellos tiene conocimiento respecto al tema lo que corresponde al 33%, no así los 14 estudiantes restantes para un 67%. Teniendo relación con la pregunta anterior, en la que el desconocimiento de la temática desfavorece una correcta evaluación de las técnicas de *watermarking* la que permiten probar la autoría de un *software* (Moruno, 2008).

La figura 2.11 referida a si conoce que en Cuba se ponen en funcionamiento algún algoritmo de *software watermarking* como contramedida frente a la piratería de *software*.

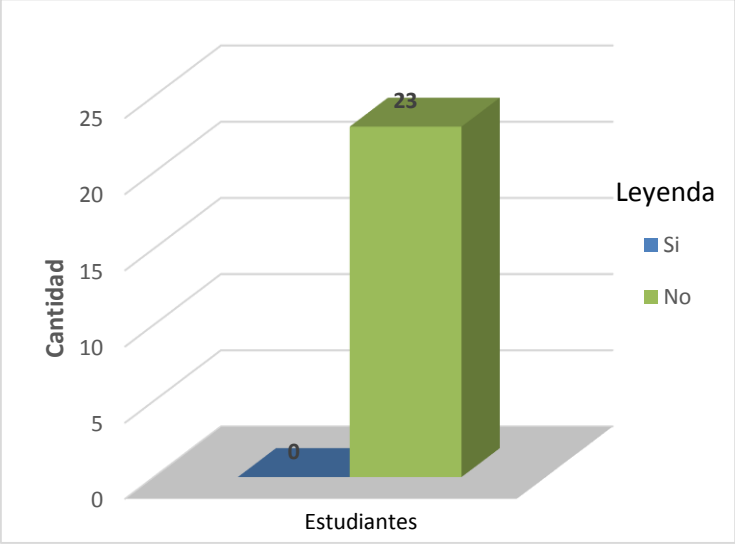

Figura 2.12 Conocimiento si Cuba pone en funcionamiento algún algoritmo de *software watermarking* como contramedida a la piratería de *software*

Fuente: Elaboración propia

El 100% de los estudiantes encuestados argumentó no tener conocimientos representando a los 21 estudiantes y evidencia la importancia del tema. Lo que demuestra la relación entre docentes y estudiantes respecto a esta cuestión, demostrando la falta del conocimiento sobre el tema. Respecto a ello, se coincide con los postulados de (Orúe, 2002) al decir que las marcas de agua para *software* son una herramienta que puede ayudar a combatir y en todo caso a entorpecer, la proliferación de los delitos informáticos.

En la figura 2.13 referida a si en la Universidad se pone en funcionamiento algún algoritmo de *software watermarking* como medida frente a la piratería de *software* y en caso de utilizar alguno nombrarlos.

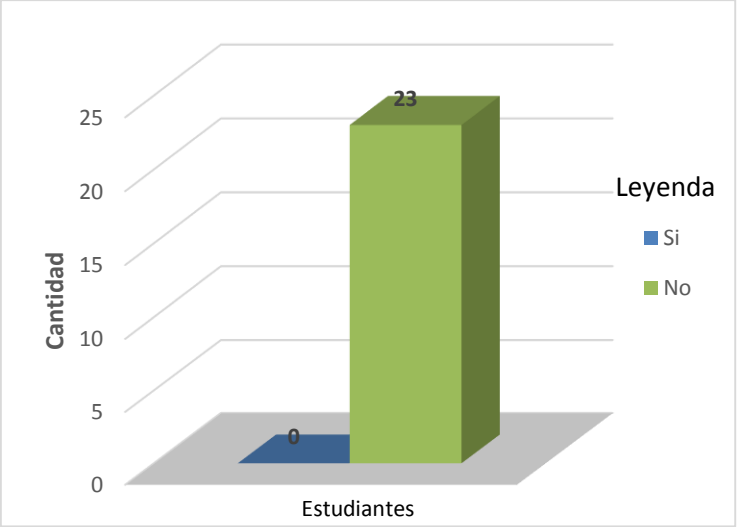

Figura 2.13 Conocimiento si en la universidad se utiliza algún algoritmo de *software watermarking* como contramedida a la piratería de *software*

Fuente: Elaboración propia

Los 21 estudiantes encuestados respondieron en su totalidad para un 100% de que no tenían conocimiento de ningún algoritmo para este fin. Este resultado concuerda con el obtenido por los docentes, el cual es necesario tener conocimiento de técnicas de marcas de agua para *software* para de esta forma poder diagnosticar cual es el algoritmo más adecuado. Además, se coincide con (Moruno, 2008) al decir que es necesario tener varios aspectos a tener en cuenta a la hora de utilizar un algoritmo de *software watermarking* las cuales son: tasa de datos requerida, forma del programa que lo contiene y amenazas esperadas.

La interrogante que hace alusión al conocimiento de los algoritmos de marcas de agua para *software* QP, QPS, QPI, SHQK, SHQK2, CT, *Threading Software watermarking*, los 21 estudiantes encuestados que representan el 100% no contestó las preguntas, por lo

que se muestra la estrecha relación entre los profesores y estudiantes al evidenciarse que el desconocimiento que existe en el claustro de profesores sobre dicha temática, que influye en la formación general con que se gradúan los mismos. El desarrollo rápido del *internet* propicia la copia y distribución de contenidos digitales, favoreciendo la piratería de *software*. Por lo que se debe estar actualizado sobre lo que investiga la comunidad científica respecto al tema (Zhu, 2007). Se coincide con el criterio del autor antes expuesto ya que el desfavorable conocimiento respecto a los algoritmos más utilizados sobre la temática contribuye a un bajo nivel de la seguridad de los *softwares* los cuales deben estar protegidos con herramientas que aseguren el derecho de autoría del mismo.

La pregunta referida a la utilización del algoritmo de marca de agua para *software* y mencionar si fue aplicado en su proyecto de tesis. Los 21 estudiantes encuestados para un 100% no respondieron la pregunta, lo que pone de manifiesto la estrecha relación con la pregunta anterior y los resultados obtenidos en la encuesta aplicada a profesores.

2.4. Propuesta del algoritmo de marcas de agua para *software*

Para que un algoritmo de marca de agua para *software* sea de mayor utilidad se debe combinar con técnicas criptográficas, de modo que el mensaje a ocultar debe cifrarse robustamente y luego ser introducido en el objeto contenedor. De este modo al interceptor le resultaría difícil reconocer el *watermark* y por lógica llegar a obtener el mensaje a ocultar (Riverón, 2014).

Esquema de compartición de secretos (esquema de Shamir)

El esquema de Shamir divide un secreto en n partes, que se reparten entre n participantes. La finalidad es que la presencia de k de estos participantes $(k \leq n)$ sea necesaria y suficiente para reconstruir el secreto (Tena, 2010).

Definición de esquema de umbral:

Para un par de valores k y n, $k < n$, un (k,n) esquema umbral consiste en particiones $D_1, D_2, \ldots, D_n \in Z/p$ y un valor secreto $D \in Z/p$ tal que:

1) El conocimiento de k, o más, participaciones D_i es suficiente para calcular la clave secreta D.
2) El conocimiento de $k-1$, o menos participaciones, deja la clave D completamente indeterminada.

El esquema de Shamir trabaja con polinomios donde el valor secreto será el término independiente de un polinomio. Dicho esquema se basa en el hecho de que un polinomio de grado $k-1$ se puede caracterizar por sus k coeficientes o bien dando valores en k, puntos diferentes (Parakh and Subhash, 2011).

El valor secreto a guardar es $D = a_0 \in Z/p$ y que $a_1, a_2, \ldots a_{k-1} \in Z/p$ son valores escogidos al azar, los cuales son coeficientes del polinomio:

$$A(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} \in Z/p$$

Considerando n elementos diferentes $x_1, x_2, \ldots, x_n \in Z/p$ y se calcula, para cada uno de ellos $A(x_i) = D_i$

La participación que se dará al i-ésimo participante es D_i.

Cualquier grupo de k o más participantes permite reconstruir el polinomio, aplicando el teorema de interpolación de Lagrange, al polinomio $A(x)$ y, por lo tanto, calcular la clave

$$D = a_0.$$

Sean $\{i_1, \cdots, i_k\}$ los k participantes, entonces:

$A(x) = \sum_{i_j=i_1}^{i_k} D_{i_j} . q_{i_j}(x)$ donde: $q_{i_j}(x) = \prod_{i_t=i_j}^{i_k} \frac{x-x_{i_t}}{x_{i_j}-x_{i_t}}$ de ahí se puede encontrar el secreto $D = a_0$

2.5. Diseño de un algoritmo de *software watermarking*

Antes de diseñar un algoritmo hay que tener en cuenta tres aspectos principales los cuales son (Tomas, 2006):

- Tasa de datos
- Forma en que está dentro del programa
- Tipos de ataques esperados.

En la presente investigación el *watermark* va estar distribuido en todo el programa, en los bloques básicos de ejecución de un programa, siendo parte esencial de su funcionamiento, lo cual permitirá adquirir cualquier tamaño por la capacidad de fraccionar el secreto, además de brindar en caso de alguna alteración inadecuada que el *software* quede inutilizable. El mismo es basado en el esquema de Shamir, otro aspecto a considerar son los tipos de amenazas que se le puede realizar al *watermark* el cual debe ser resistente a los ataques por deformación, adición, confabulación y sustracción.

El diseño de este algoritmo de marca de agua para software está constituido en dos fases las cuales van a posibilitar la inserción y la extracción del *watermark*. La fase de inserción consta de dos partes, al igual que la fase de extracción. A continuación, se exponen ambas fases:

Fase de inserción del *watermark*

1. Se debe construir un polinomio de grado n, en el cual el término independiente es el secreto a distribuir, se calcula n puntos del polinomio en los cuales va estar implícitamente el secreto, siempre manteniendo un esquema de umbral.

2. Se distribuye los puntos calculados en un grafo de control de flujo de un programa, principalmente en las funciones esenciales del mismo. Lo que posibilita de esta manera que en caso de alguna alteración del nodo este pueda quedar inutilizable, además que, por esta forma de fraccionar el secreto, el mismo pase inadvertido a la vista de un atacante, brindando la imperceptibilidad de él.

A continuación, se expone un diagrama del proceso de inserción del *watermark* en un programa:

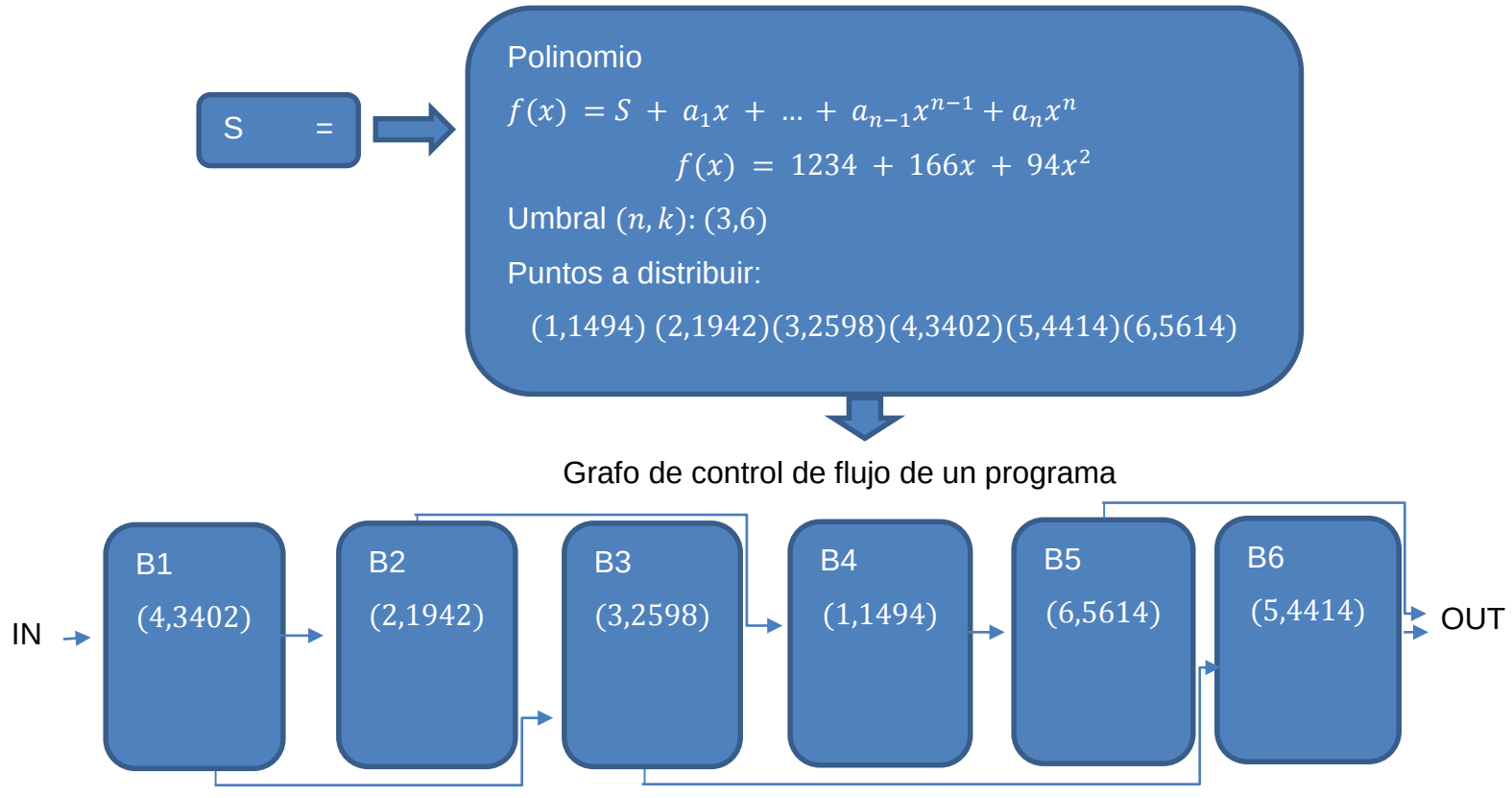

Figura 2.14 Diagrama de inserción del *watermark*

Fuente: Elaboración propia

En el grafo de control de flujo de un programa: B1…B6 significa los bloques del grafo donde se van a distribuir de forma aleatoria los puntos obtenidos, pero principalmente se distribuirán en las funciones principales del programa. De esta forma se puede recuperar

caso de ser atacado uno o varios bloques del grafo obtener el secreto manteniendo el mínimo de bloques íntegros basándose en el esquema de umbral antes definido.

Fase de extracción del *watermark*

1. Manteniendo el esquema de umbral (k, n) donde k es el número mínimos de puntos necesarios para poder obtener el secreto y n la cantidad de puntos a evaluar.
2. Obtener la cantidad de puntos mínimos para poder acceder al secreto mediante el teorema de Langrage, de esta forma se regresa al polinomio original en el cual el término independiente es el secreto.

En el siguiente, diagrama muestra el proceso de extracción del *watermark*:

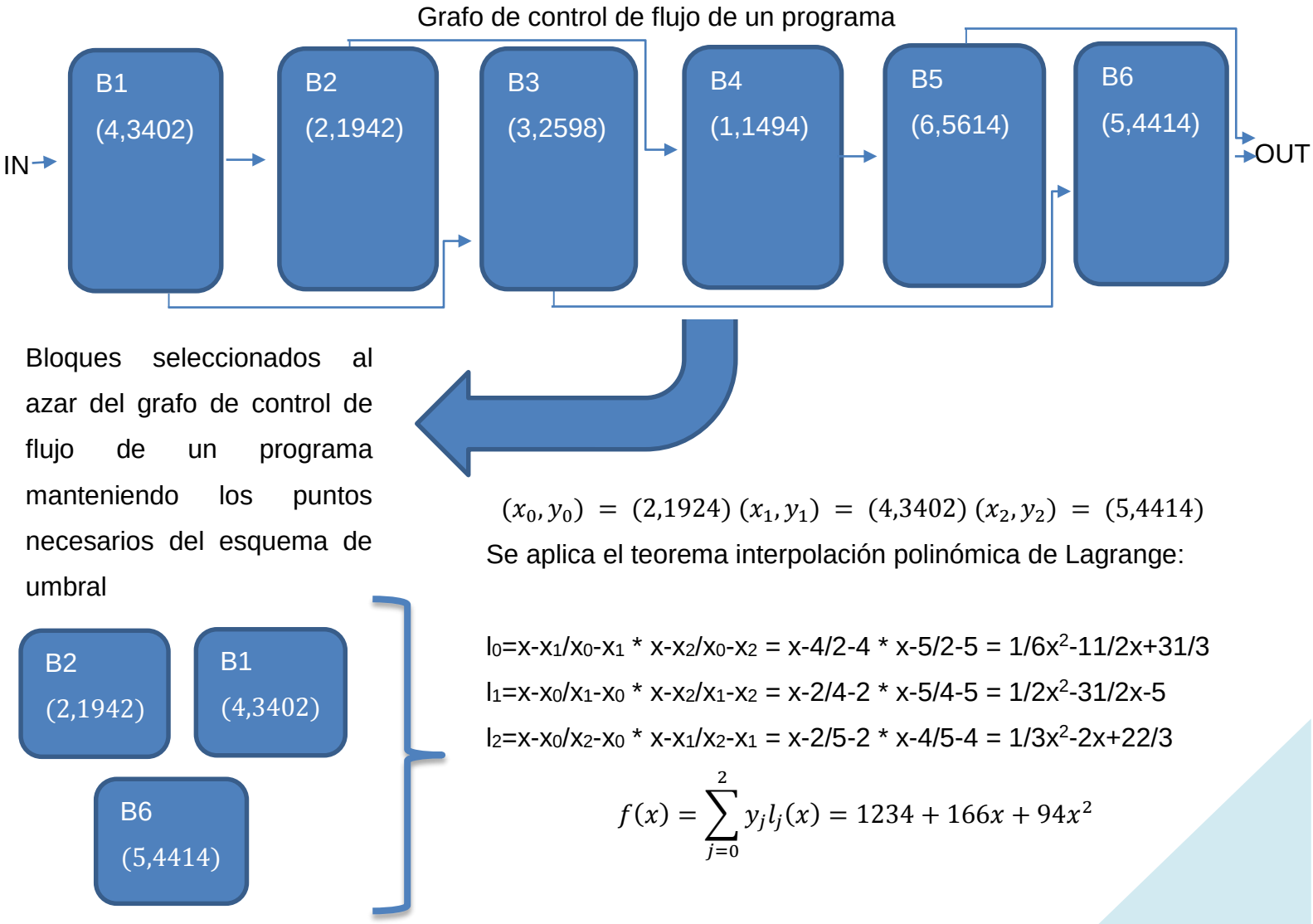

Figura 2.14 Diagrama de inserción del *watermark*

Fuente: Elaboración propia

Un aspecto a tener en cuenta de este algoritmo son los posibles tipos de ataques los cuales puede ser resistente:

- Sustracción
- Adicción
- Deformación

estos ataques resultarían fallidos producto a que se fracciona la marca y se distribuye por los nodos del grafo de control de flujo de un programa haciendo de esta manera que un atacante pueda modificar uno o varios nodos sin obtener su propósito, ya que el algoritmo se basa en el esquema de Shamir.

Este algoritmo permite que en el caso de alterar algún nodo del grafo de control de flujo en el cual están distribuidos los puntos que, el programa pueda quedar inutilizable producto a que los puntos distribuidos van a pertenecer a las principales funciones del bloque del grafo, además de que con una cantidad mínima de puntos se puede obtener el polinomio original y por consiguiente el secreto distribuido.

Se basó en el esquema de Shamir el cual es un algoritmo de criptografía, el cual brinda robustez, al decir Wagner en el 2004 cuando plantea que el esquema brinda las siguientes propiedades dándole fortaleza al algoritmo.

Es perfecto: si conocemos $t-1$ o menos fragmentos, la probabilidad de encontrar el valor para el secreto compartido en el campo Z/p, es siempre la misma.

Es ideal: Como los fragmentos y el secreto pertenecen al campo Z/p, entonces poseen la misma longitud en *bits*.

Es escalable: Es posible expandir para nuevos usuarios en este caso distribuirlos en varios nodos, esto significa que podemos calcular y distribuir nuevos fragmentos. Por otro lado, se observa que si se ataca a un nodo el cual contiene un fragmento, se puede calcular el secreto sin importar el nodo el cual fue atacado.

Es seguro: La seguridad del esquema de Shamir se basa principalmente en la elección del umbral.

2.6. Implementación del algoritmo de marca de agua para *software*

Para la implementación de este algoritmo se utilizó como lenguaje de programación Java, es un lenguaje neutral, portable, robusto, estable, independiente de la plataforma, sencillo y de alto nivel. Tiene las ventajas de un lenguaje de programación orientado a objetos y es flexible para su integración con gran cantidad de tecnologías, permite la ejecución de las aplicaciones en múltiples plataformas de *hardware* y sistemas operativos (Unix, Linux, OS/390, Windows entre otros sistemas operativos). Esto se debe a su ambiente JRE (*Java Runtime Environment*) construido sobre la máquina virtual java JVM (*Java Virtual Machine*) que rompe la dependencia entre el código compilado y cualquier plataforma de *hardware* o sistema operativo en particular (Letelier,2006)

Entre las ventajas de Java con respecto a otros lenguajes de programación se pueden citar:

- Las aplicaciones Java pueden correr en una amplia gama de sistemas operativos y de arquitecturas hardware.
- La tecnología Java es una tecnología abierta y se basa en gran parte en estándares de organizaciones de normalización y estándares empresariales.

Java goza ya de una cierta veteranía y ha probado su eficacia en muchos entornos y situaciones empresariales distintas. (Letelier,2006)

El JDK (siglas del inglés *Java Development Kit*) es un entorno de desarrollo conformado por un conjunto de programas y librerías que permiten desarrollar, compilar, probar y ejecutar aplicaciones, empleando el lenguaje de programación Java. Existen varias versiones de este, pero ha sido utilizada en esta investigación la versión 1.6.

El entorno de desarrollo utilizado fue el *NetBeans,* el mismo es hecho para el lenguaje de programación Java. Este es un IDE gratuito sin restricciones de uso.

La aplicación que se propone consta de dos fases, una de inserción y otra de extracción. La primera fase está constituida por el grado del polinomio, los coeficientes a utilizar, así como la cantidad de puntos a distribuir, además del mensaje a codificar. En la fase de extracción, se ingresan los puntos necesarios para obtener el mensaje codificado basado en el esquema de Shamir.

Fase de inserción:

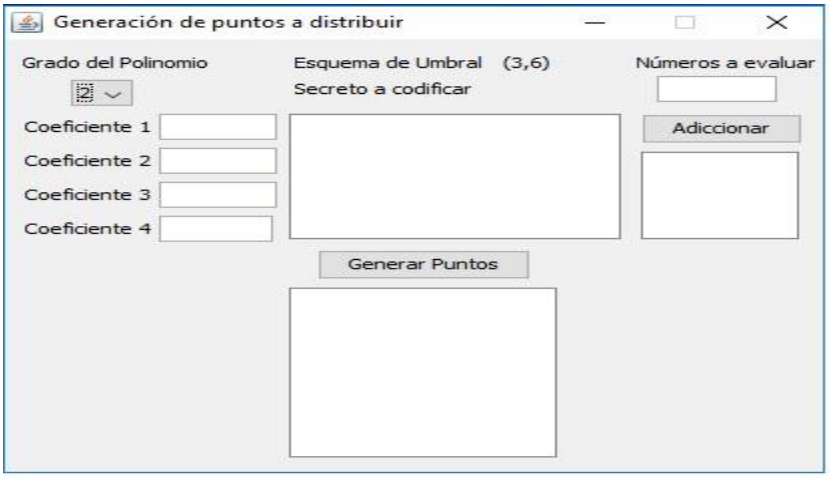

Figura. 2.15. Generación de puntos a distribuir

En la ventana de generación de puntos a distribuir en la parte superior izquierda, se puede seleccionar el grado del polinomio, que brinda el esquema del umbral, el cual dice los

puntos necesarios para poder generar el polinomio y los puntos mínimos a distribuir. Una vez realizado esto se llenan los campos del coeficiente en dependencia del grado del polinomio, se adicionan los números a evaluar en el polinomio, el secreto a codificar el que debe ser un número. Posteriormente, se le da click al botón generar puntos donde se mostrará una secuencia de puntos a distribuir.

Ejemplo de generación de puntos a distribuir donde el primer coeficiente tiene valor 26, el segundo 35 el secreto es 6580, la cantidad de números a evaluar son los siguientes: 20, 13, 15, 1, 2, 3. Los puntos generados son: (20,17680), (13,11429), (15,12955), (1,6641), (2,6754), (3,6919)

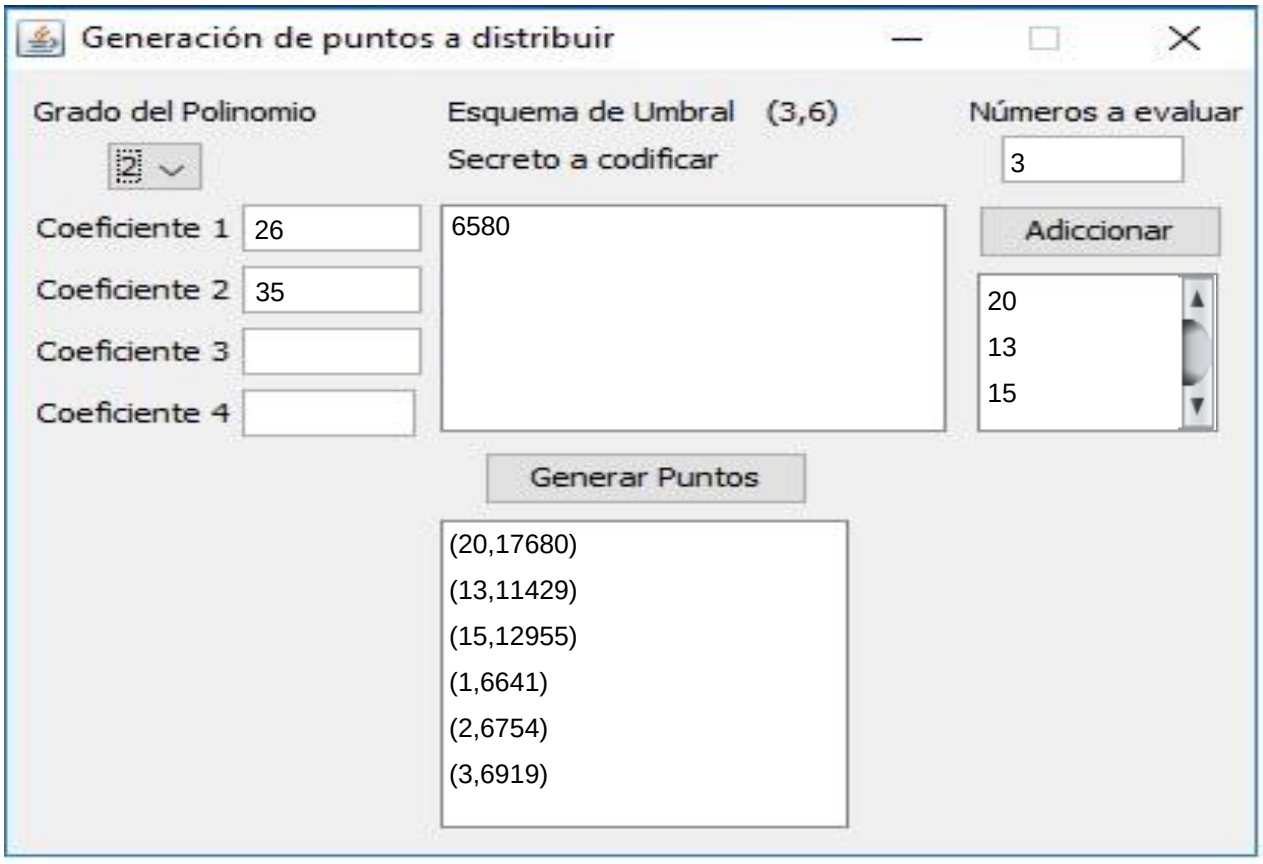

Figura. 2.16 a. Ejemplo de generación de puntos a distribuir

Ejemplo de generación de puntos a distribuir de la figura. 2.16 b. donde el primer coeficiente tiene valor 2, el segundo 3, el tercero 5 y el último 1; el secreto es 201389, la

cantidad de números a evaluar son los siguientes: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Los puntos generados son: (1,201394), (2, 201427), (3,201560), (4,201913), (5,202654), (6,203999), (7, 206212), (8, 209605), (9, 24538), (10, 221419).

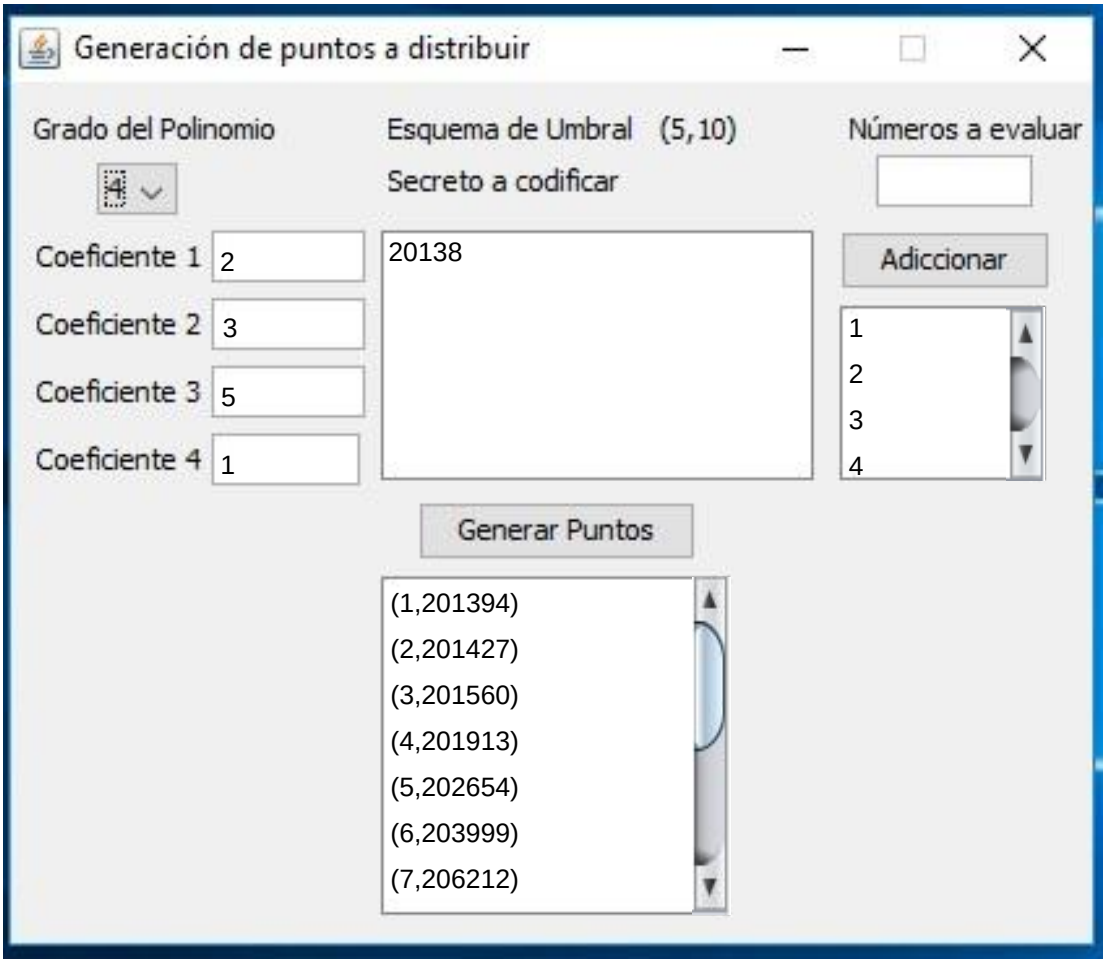

Figura. 2.16 b. Ejemplo de generación de puntos a distribuir

Fase de extracción:

Figura. 2.17. Generación de secreto

En la ventana de generar el secreto se ingresan los puntos necesarios para poder obtener nuevamente el secreto. Una vez ingresado estos puntos, se da click al botón generar aplica el teorema de Lagrange para volver a obtener el mensaje a codificar.

El ejemplo siguiente muestra como con los puntos necesarios basados en el esquema de umbral se puede volver a obtener el secreto introducido por primera vez.

Figura. 2.18 a. Ejemplo de generación de secreto

En la figura 2.18 b. se muestra la obtención del secreto a partir de otros puntos generados.

Figura. 2.18 b. Ejemplo de generación de secreto

CONCLUSIONES

1. La revisión de los antecedentes teóricos y metodológicos relacionados con las marcas de agua para *software* permitió identificar los autores nacionales e internacionales que trabajan dicho tema, lo que sirvió como fundamento para llevar a cabo el diagnóstico, discutir los resultados obtenidos y elaborar la propuesta.

2. El diagnóstico llevado a cabo a estudiantes y profesores de la Carrera de Ingeniería informática de la UNAH permitió constatar que:

    - De forma general no existe un conocimiento sobre el tema investigado.
    - A pesar de que los egresados de la misma deben de ser capaces de programar y dirigir a otros programadores entre otras funciones, no se tienen en cuenta los algoritmos de *software watermarking* para la protección de los productos elaborados en la Carrera de Informática de la UNAH como resultado final de su tesis de diploma.

2. El algoritmo propuesto se basó en el esquema de secreto compartido de Shamir, el cual constituye un método práctico y útil para proteger la propiedad intelectual contenida en un software. El mismo responde a los resultados del diagnóstico llevado acabo, como una necesidad para los *softwares* producidos en la Carrera de Informática de la UNAH.

3. Se elaboró una herramienta utilizando el algoritmo propuesto, la cual permite generar puntos a partir de un polinomio de orden 2, 3 y 4. Los cuales van a ser insertados en el código fuente de un software, de igual manera, a partir de estos puntos se obtiene el secreto a extraer.

RECOMENDACIONES

A partir de los resultados obtenidos en la realización de esta investigación de maestría se proponen las siguientes recomendaciones:

1. Aplicar el algoritmo de marcas de agua para *software* y validar los resultados obtenidos mediante la técnica de base subjetiva IADOV, añadiéndolo a la herramienta *Sandmark*.

2. Compartir los resultados de la investigación a otras universidades e instituciones afines del país y dar seguimiento del mismo para su evolución.

BIBLIOGRAFÍA

- Anglia Ruskin University (2011). *Guide to the Harvard Style of Referencing* [online]. Disponible: http://libweb.anglia.ac.uk. Consultado: [15 de junio del 2014].
- Areitio Bertolín, J. (2004). *Análisis y desarrollo de esquemas para la compartición e intercambio de secretos*. [s.l.s.n.s.p].
- Blakley, R. (1979). Safeguarding cryptographic keys*, Proceedings of AFIPS National Computer Conference*, vol.48*,* pp.313-317.
- Business Software Alliance. BSA (2011). *Research and statistics* [online]. Disponible: http://ww2.bsa.org/country/Research%20and%20Statistics.aspx Consultado: [3 de Agosto del 2014].
- _____ (2013). *News and Events* [online]. Disponible: http://ww2.bsa.org/country/News%20and%20Events.aspx Consultado: [27 de Septiembre del 2014].
- Castellanos Simons, B. (1998). *Nuevos escenarios, nuevos actores, nuevas estrategias*. La Habana. Instituto Superior Pedagógico “Enrique José Varona”. Facultad de Ciencias de la Educación, p.44.
- Claverol M., Chionis I., and Chroni M., Simó E., and Zaragozá M. (1999). *Matemática Discreta. Teoría de Grafos*. Departamento Matemática Aplicada IV. EPSEVG – UPC.
- Collberg, C. (2004). *Dynamic Graph-Based Software Watermarking*. Technical Report TR04-08. Department of Computer Science, University of Arizona. USA, 46p.
- _____ (2002). *Dynamic Path-Based Software Watermarking*. Department of Computer Science, University of Arizona, Tucson, USA, 20p.
- Collberg C., Jha S., Tomko D., and Wang H (2001). *A General Architecture for Software Watemarking*. Department of Computer Science, University of Arizona, Tucson, 35p.
- Collberg, C. and Thomborson, C. (2005). Error-Correcting Graphs for Software Watermarking. New Zealand. *New Economy Research Fund of New Zealand*. [s.p].

- _____ (1998). *On the limits of software watermarking*, in Technical Report No.164, Department of Computer Science, The University of Auckland, 25p.
- _____ (1999). *Software watermarking: Models and dynamic embeddings*, in Proceedings of Symposium on Principles of Programming Languages, pp. 311–324.
- _____ (2002). Watermarking, Tamper-Proofing, and Obfuscation -- Tools for Software Protection. University of Auckland Computer Science. *IEEE Transactions on Software Engineering,* 15p.
- Cuba. Ministerio de Justicia (2008). Ley 14 de Derecho de Autor. La Habana. Ministerio de Justicia. [en línea]. Disponible en: www.gacetaoficial.cu. Consultado: [10 de diciembre del 2014].
- Davidson R., and Myhrvold N. (1996). *Method and system for generating and auditing a signature for a computer program*. US Patent 5,559,884, September 1996. Assignee: Microsoft Corporation.
- Eilam, E. (2005). *Reversing - Secrets of Reverse Engineering*. Indianapolis, Indiana. Wiley Published Inc., p. 589
- Fox, B. (2007). *Pirates keep pirating*. [s.l.]. *New Scientist,* p. 194.
- Fabien A., Petitcolas, R. (2011). *Information Hiding A Survey*. [en línea]. Disponible en: www.ieeexplore.com. Consultado: [20 de abril del 2015].
- Fuente O'Connor J. (2014). *Funciones de interpolación y aproximación.* Universidad Politécnica de Madrid. Escuela Técnica Superior de Ingenieros Industriales, 35p.
- Gerardino M. (2007). *Técnicas Para La Transmisión De Mensajes Ocultos, Criptografia, Esteganografia Y Watermarking. Simposio Internacional De Telecomunicaciones*, *Mérida-Venezuela*.
- Ginger Myles, C. (2003). *Software Watermarking Through Register Allocation: Implementation, Analysis, and Attacks*. University of Arizona. Department of Computer Science, [s.p].

- Goel S. (2008). *Watermarking & Steganography.* University at Albany, SUNY. [s.l.s.n.s.p].
- Gómez Cárdenas R. (2009).*Estenografía.* Tesis de Maestría en Seguridad Informática, ITESM-CEM, h.25.
- Gopal, R., Sanders, G. (1998). International Software Piracy: Analysis of Key Issuesand Impacts. (s.l.). *Information Systems Research,* pp. 380-397.
- Grover D. (1997). T*he Protection of Computer Software - Its Technology and Applications*, 2nd ed. Cambridge University Press.
- Hachez G. (2003). *A comparative study of software protection tools suited for ecommerce with contributions to software watermarking and smart cards*, PhD thesis. Universite Catholique de Louvain.
- Hardikkumar Desai V. (2012). Steganography, Cryptography, Watermarking: A Comparative Study. *Journal of Global Research in Computer Science,* Volume 3, No. 12, Research Scholar, Singhania University, 10p.
- Hernández, N. (2013).*Estrategia para la Gestión de La Propiedad Industrial en la Universidad Agraria de La Habana.* Tesis de Maestría. Centro de Estudios de Desarrollo Agrario y Rural, h.14.
- Holmes K. (1994).*Computer software protection. US Patent 5,287,407*, Assignee: International Buisness Machines.
- Johnsonbaugh R. (1999).*Matemáticas discretas*.[s.l.] Prentice Hall, 400p.
- Letelier, P., (2006). *Metodologías ágiles para el desarrollo de software*: *Extreme Programming (XP).*
- López, J. (2008). *Criptografía y Seguridad en Computadores*. España.
- Lu, J. (2009). *Chinese culture and software copyright*. [s.l.]. New Media & Amp Society, pp. 1372-1393.
- Mathews J. (2000). Métodos numéricos con Matlab. Prentice Hall.
- Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii. (2000). *A practical method for watermarking Java programs. In compsac2000*, 24th Computer Software and Applications Conference, [s.p.].

- Moreno, M. and Horta, E. (2007). *Selección de lecturas de Propiedad Industrial*. Tomo I, II. La Habana. Félix Varela.
- Moruno, M. (2008). *Estudio de técnicas de inserción de marcas de agua sobre software* (*Software watermarking*). UPC. España, 157p.
- Moskowitz S and Cooperman M. (1994). *Method for stega-cipher protection of computer code*.US Patent. vol. 5, pp. 745,569.
- Myles, G., Collberg, C., Heidepriem, Z. (2005). *The evaluation of two software watermarking algorithms*. [s.l.s.n.], pp. 923-938.
- Nagra, J. (2005). *Functional Taxonomy for Software Watermarking. Australasian Computer Science Conference*. Melbourne, Australia.
- _____ (2006). *Software Watermarking: Protective Terminology*. [s.l.s.n.s.p].
- _____ (2007). *Threading Software Watermarks*. Tesis of PhD. The University of Auckland.
- Nagra, J, Thomborson C., and Collberg C. (2002). *Software watermarking: Protective terminology*, in Proceedings of the ACSC.
- Núñez, J. (2010). *Conocimiento académico y sociedad. Ensayos sobre política universitaria de investigación y postgrado*. La Habana. Universidad de La Habana.
- Nuñez Y. (2013). *Modelo no determinista para la Auto - verificación de integridad de componentes de software*. Alicante, España, 255p.
- Orúe López, B. (2002). *Boletín del Criptonomicón* [en línea]. Disponible en http://www.iec.csic.es/criptonomicon/copyright.html [Consultado: 20 junio del 2014].
- Organización Mundial de Propiedad Intelectual. OMPI. (2015). *Principios Básicos de la Propiedad Intelectual* [en línea]. Disponible en: www.wipo.int/ebookshop.html [Consulta: 10 enero 2015].
- Parakh, A. andSubhash, K. (2011). *Space efficient secret sharing, Information Sciences*. [s.l.s.n.s.p].
- Qu G. and Potkonjak M. (1998). Analysis of watermarking techniques for graph coloringproblem, in*: IEEE/ACM International Conference on Computer Aided Design*, pp. 190–193.

- Riverón, L. (2014). *Solución para la protección de la información basada en esteganografía.* La Habana. Instituto Superior Politécnico “José Antonio Echeverría”.
- Stern P., Hachez G., Koeune F., and Quisquater J. (1999). *Robust object watermarking: Application to code. In Information Hiding*, [s.l.s.n], pp. 368.378.
- Taboada, A. (2010). *Modelo integrado de gestión de la ciencia, la innovación tecnológica y el conocimiento para la Universidad Agraria de La Habana*. Tesis de Doctorado. Cuba, Universidad de Pinar del Río.
- Tapas Ranjan S. and Collberg C. (2004). Software watermarking in the frequency domain: implementation, analysis, and attacks. *Journal of Computer Security*, Volume 13 Issue 5.
- Tena Ayuso, J. (2010). *Protocolos criptográficos.* Universitat Oberta de Catalunya. [s.l.s.n.s.p].
- Tomas Ciruana J. (2006). *Watermarking de Software: Estado del arte.* Barcelona. España.Departamento de Ingeniería Telemática (UPC).
- Van, H., Hogenbirk, A. (2005). Multimedia, Entertainment, and Business Software Copyright Piracy: A Cross-National Study. (s.l.). *Journal of Media Economics*. p. 109-129.
- Vázquez González, L. (2004). *Métodos Computacionales para Esquemas de Compartición de Secretos Ideales.* México, DF. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional de México.
- Vela, J. (2000). Educación Superior: inversión para el futuro. Universidad de la Habana. Conferencia impartida en el 50 aniversario de la Unión Latinoamericana de Universidades (UDUAL). *Revista Cubana Educación Media Superior* 14 (2): 171-183.
- Venkatesan R., Vazirani V., and Sinha S. (2001). *A graph theoretic approach to software watermarking*. In 4th International Information Hiding Workshop, Pittsburgh, PA.
- Wagner D. (2004). Crytography, Spring. [s.l.s.n.s.p].

- Zhu W. (2007). *Concepts and Techniques in Software Watermarking and Obfuscation*. New Zealand. The Department of Computer Sciences, The University of Auckland.
- Zhu W., Thomborson C., and Wang F. (2005). Survey of Software Watermarking. *New Economy Research Fund of New Zealand,* pp. 454–458.

GLOSARIO DE TÉRMINOS

BSA – Business Software Alliance, Alianza de negocios informáticos

CFG – Grafo de control de flujo

CRD– Curso regular diurno

CPE – curso por encuentro

e-commercey – Comercio electrónico

GTW – Graph Theoretic Watermarking

IBM–International Business Machines Corporation, Corporación de internacional de negocios de maquinas

IDC – Internacional Data Corporation, Corporación internacional de datos

JPEG – Joint Photographic Experts Group, Grupo conjunto de expertos en fotografía

MD5 –Message-Digest Algorithm 5, Algoritmo de Resumen del Mensaje 5

OMPI –Organización Mundial de la Propiedad Intelectual

SAD-TE –Ingeniero en Sistemas Automatizados de Dirección Técnico Económico

TIC–Tecnologías de la Información y las Comunicaciones

UNAH –Universidad Agraria de La Habana

ANEXO 1

ENCUESTA A PROFESORES

Estimado profesor:

Estamos llevando a cabo una investigación con el propósito de diagnosticar si se tiene en cuenta en el proceso de elaboración de un *software*, algún algoritmo que permita proteger la propiedad intelectual del mismo. Por lo que recabamos de su colaboración en aras de evitar la piratería.

1. ¿Usted tiene conocimiento de la existencia en Cuba de alguna ley que proteja la propiedad intelectual de un *software*?

   Si____ No___ No se____

2. ¿Tiene dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*)?

   Si____ No___ No se____

3. ¿Conoce en que consiste las marcas de agua para *software (software watermarking*)?

   Si____ No___ No se____

4. ¿Tiene conocimiento acerca de los tipos de ataques que se les puede realizar al *software watermarking*?

   Si____ No___ No se____

5. ¿Conoce usted si en Cuba se ponen en funcionamiento algún algoritmo de *software watermarking* como contramedida frente a la piratería de *software*?

   Si____ No___ No se____

6. ¿Conoce usted si en la universidad se ponen en funcionamiento algún algoritmo de *software watermarking* como medida frente a la piratería de *software*, diga cuál?

   Si____ No___ No se____ Nombre del algoritmo _____

7. Marque con una (x) de los siguientes algoritmos de *software watermarking* cuales usted conoce:
   QP__, QPS__, QPI___, SHQK___, SHQK2__, CT__,
   Threading Software Watermarking___,

8. En caso de utilizar algún algoritmo diga en que *software* fue aplicado y si cumplió con sus expectativas.

   _____

   _____

ANEXO 2

ENCUESTA A ESTUDIANTES

Estimado estudiante:

Estamos llevando a cabo una investigación con el propósito de diagnosticar si se tiene en cuenta en el proceso de elaboración de un *software*, algún algoritmo que permita proteger la propiedad intelectual del mismo. Por lo que recabamos de su colaboración en aras de evitar la piratería.

1. ¿Usted tiene conocimiento de la existencia en Cuba de alguna ley que proteja la propiedad intelectual de un *software*?.
   Si____ No___ No se____

2. ¿Tiene dominio sobre los ataques que se le puede realizar a la propiedad intelectual contenida en un *software* (ingeniería inversa, la piratería, y el *tampering*)?

   Si____ No___ No se____

3. ¿Conoce en que consiste las marcas de agua para *software* (*software watermarking*)?
   Si____ No___ No se____

4. ¿Tiene conocimiento acerca de los tipos de ataques que se les puede realizar al *software watermarking*?

   Si____ No___ No se____

5. ¿Conoce usted si en Cuba se ponen en funcionamiento algún algoritmo de *software watermarking* como contramedida frente a la piratería de *software*?.

   Si____ No___ No se____

6. ¿Conoce usted si en la universidad se ponen en funcionamiento algún algoritmo de *software watermarking* como medida frente a la piratería de *software*, diga cuál?.

   Si____ No___ No se____ Nombre del algoritmo _____

7. Marque con una (x) de los siguientes algoritmos de *software watermarking* cuales usted conoce:
   QP__, QPS__, QPI___, SHQK___, SHQK2__, CT__,
   Threading Software Watermarking___,

8. En caso de utilizar algún algoritmo diga si fue aplicado en su proyecto de tesis y si cumplió con sus expectativas.

_____

_____