UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA INFORMÁTICA Y SISTEMAS

ALGORITMO DE OFUSCACIÓN PARA PROTECCIÓN DEL CÓDIGO FUENTE PHP COMO PROPIEDAD INTELECTUAL Y EL NIVEL DE SATISFACCIÓN DE LOS DESARROLLADORES DE SOFTWARE EN LA REGIÓN APURÍMAC – 2012

TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO Y SISTEMAS

FLOR CAGNIY CÁRDENAS MARIÑO

Abancay, Diciembre del 2013
PERÚ

UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y SISTEMAS

Jurado Calificador Integrado Por:

M.Sc Ecler Mamani Vilca
PRESIDENTE

M.Sc. Manuel Jesús Ibarra Cabrera
Jurado

Lic. Marco Antonio Ramos Alva
Jurado

M.Sc. Hugo David Calderón Vilca
Asesor

ALGORITMO DE OFUSCACIÓN PARA PROTECCIÓN DEL CÓDIGO FUENTE PHP COMO PROPIEDAD INTELECTUAL Y EL NIVEL DE SATISFACCIÓN DE LOS DESARROLLADORES DE SOFTWARE EN LA REGIÓN APURÍMAC – 2012.

DEDICATORIA

A Dios por que cada instante nos guía, nos ilumina para cumplir con nuestras metas.

A mi familia (David, David, Angie) por el apoyo incondicional, que me brindan para mi realización profesional.

A mis padres Víctor y Victoria, que siempre están ahí para darme su apoyo incondicional y sus consejos, a mis hermanos por su apoyo, motivación y para la ejecución de la presente investigación.

AGRADECIMIENTOS

A todos los que contribuyeron para la

realización de esta investigación.

ÍNDICE

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El motivo de esta investigación fue que los desarrolladores de software que se basan en las tecnologías LAMP, al implementar sistemas de información se ven obligados a dejar el código fuente PHP, perdiendo de esta manera la propiedad intelectual, dando riesgo de que sean copiadas y reutilizadas por terceros no autorizados.

Por lo que, la presente investigación ha tenido como objetivo la creación de un algoritmo de ofuscación de código fuente PHP el cual contribuye con la protección intelectual de los desarrolladores que se basan en la tecnología LAMP en la Región Apurímac. Cuya prueba de hipótesis planteada con la distribución binomial P calculada es 0,00462 mayor a la tabla permitiendo aceptar H1 apoyando de esta manera que el algoritmo de ofuscación protege el código fuente PHP como propiedad intelectual de los desarrolladores de la Región Apurímac. Así mismo para el nivel de satisfacción con chi-cuadrado = 3,52 menor $x^2 tabla = 9{,}488$, concluyendo que el nivel de satisfacción es aceptable.

Para el diseño del algoritmo se ha utilizado el método de la programación dinámica, la cual ha permitido dividir el problema original en subproblemas más simples para que sea más fácil el diseño del algoritmo; la estrategia tomada para la programación del algoritmo fue el paradigma de la programación imperativa y la técnica de ofuscación que se ha optado es la ofuscación de datos.

El algoritmo creado ofusca el código fuente PHP, cuyo funcionamiento se refleja de la siguiente manera: Recibe como entrada el sistema desarrollado en PHP, en el

proceso ofusca todo los archivos .php sin alterar su funcionamiento y deja sin modificaciones el resto de los archivos, finalmente da como salida el sistema ofuscado.

Finalmente se hizo pruebas de ofuscación con varias iteraciones a un software, de manera que se pudo probar que cuanto más iteraciones tenga el algoritmo más protegida estará el código fuente PHP, así se evitará la modificación no autorizada en la programación de manera que el desarrollador podrá exponer y poner a disposición de terceros el software como creación intelectual bajo las condiciones que viere por conveniente con licencias de uso ya sea con costo o no. Así mismo se ha realizado un test a los desarrolladores acerca del nivel de satisfacción demostrándose que es aceptable.

INTRODUCCIÓN

La presente investigación titulado "ALGORITMO DE OFUSCACIÓN PARA PROTECCIÓN DEL CÓDIGO FUENTE PHP COMO PROPIEDAD INTELECTUAL Y EL NIVEL DE SATISFACCIÓN DE LOS DESARROLLADORES DE SOFTWARE DE LA REGIÓN APURÍMAC- 2012", cuyo objetivo es proteger el código fuente PHP como propiedad intelectual de los desarrolladores de software en lenguaje de programación PHP, a partir de este objetivo general se determinó los siguientes objetivos específicos; crear el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012 y determinar el nivel de satisfacción de los desarrolladores de software al usar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

En esta parte de ofuscación de código fuente PHP como protección intelectual, se tomó en cuenta a los desarrolladores que utilizan las tecnologías LAMP, ya que al implementar softwares en las instituciones se ven obligados a dejar el código fuente PHP, de tal manera que se va perdiendo la propiedad intelectual, ocasionando el riesgo de que sean copiadas y reutilizadas por terceros no autorizados y generando disconformidad por parte de los programadores.

Por lo cual la creación de un algoritmo de ofuscación protege el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac, permitiendo la distribución de aplicaciones PHP sin mostrar su código

fuente y con opciones de licencia para el uso, herramienta muy necesaria para los desarrolladores de software.

El algoritmo de ofuscación de código fuente PHP ayudará a evitar la alteración del código fuente inicial, de esta manera, el desarrollador que utiliza la tecnología LAMP puede exponer y poner a disposición de terceros el software como creación intelectual bajo las condiciones que viere por conveniente con licencias de uso ya sea con costo o no.

En el capítulo I se presenta el planteamiento y formulación del problema de la investigación, justificación y objetivos de la investigación; en el capítulo II los antecedentes de la investigación y el marco referencial; en el capítulo III se presentan las hipótesis y variables; en el capítulo IV está la metodología usada para esta investigación; en el capítulo V se presentan los resultados y discusión; finalmente se presentan las conclusiones y las recomendaciones de la investigación.

CAPÍTULO I

1. EL PROBLEMA DE INVESTIGACIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

Actualmente toda organización pública o privada requiere y/o implementan sus sistemas de información acorde a las Tecnologías de la Información y Comunicación actuales; dichos sistemas de información permiten automatizar y agilizar sus procesos internos. Muchas de estas organizaciones optan por las tecnologías LAMP (Linux como sistema operativo, Apache como servidor web, MySQL como gestor de bases de datos, PHP como Lenguaje de Programación Interpretado), esto debido a que la potencia, estabilidad, gratuidad, modificabilidad y portabilidad de Linux lo hacen el sistema operativo robusto y tiene una posición líder en el ámbito de servidores, además de la amplia variedad de protocolos de red soportados en el núcleo: IPv4, IPv6, AX.25, X.25, IPX, PPP, DDP, Netrom, Appletalk, Netware, etc..Incluyendo la mayoría de utilidades necesarias para

montar un servidor en Internet (telnet, ssh, apache, mysql, php, ftp, news, irc, etc.).

El lenguaje de Programación Interpretado PHP con "Licencia PHP" es software libre, no compatible con GPL y no es Copyleft, de manera que PHP está diseñada para incentivar la distribución del código fuente, sin embargo permite la redistribución del contenido licenciado en forma de código fuente o binaria bajo las condiciones de su licencia.

Los desarrolladores de software de la Región Apurímac, que se basan en las tecnologías LAMP, se ven obligados a dejar el código fuente en el servidor, perdiendo de esta manera la propiedad intelectual, dando riesgo de que sean copiadas y reutilizadas por terceros no autorizados, y que además pueden ocasionar pérdidas económicas para el desarrollador.

La pérdida de la propiedad intelectual también está relacionada a las pérdidas económicas para el desarrollador, ya que al no estar protegido el código fuente, otras terceras personas podrían copiar y redistribuir con costo el producto, de esta manera el desarrollador se ve afectado por la piratería del software.

El no poder proteger el código fuente de los sistemas información basados en PHP, es un vacío y un hecho por resolver para los desarrolladores de software de la Región Apurímac.

Problema general

¿En qué medida el algoritmo de ofuscación protegerá el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012?

Problema Específico

¿El algoritmo de ofuscación protegerá el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012?

¿Cuál es el nivel de satisfacción de los desarrolladores de software al usar el algoritmo de ofuscación que protege el código fuente PHP como propiedad intelectual de los desarrolladores en la Región Apurímac 2012?.

1.2. JUSTIFICACIÓN

La creación de un algoritmo de ofuscación de códigos fuente PHP, contribuye con la protección intelectual de los desarrolladores de software basado en PHP en la Región Apurímac, permitiendo la distribución de aplicaciones PHP ofuscado sin alterar su funcionalidad inicial y con opciones de licencia para el uso, herramienta muy necesaria para los desarrolladores de software.

Además, el algoritmo de ofuscación evita la modificación no autorizada del código fuente inicial del software, permitiendo así al desarrollador exponer y poner a disposición de terceros el software como su creación intelectual

bajo las condiciones que viere por conveniente con licencias de uso ya sea con costo o no.

1.3. OBJETIVOS

a) Objetivo General

Proteger el código fuente PHP como propiedad intelectual de los desarrolladores de software con la creación de un algoritmo de ofuscación en la Región Apurímac 2012.

b) Objetivos Específicos

- Crear el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012.
- Determinar el nivel de satisfacción de los desarrolladores de software al usar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

CAPÍTULO II

2. MARCO TEÓRICO

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

a) Internacional

Dolz y Parra (2006), en su trabajo de investigación "Ofuscadores de Código Intermedio. Reporte Preliminar", Universidad Nacional de Comahue - Buenos Aires-Argentina; analiza el riesgo de pérdida de propiedad intelectual que apareja esta arquitectura y se describe un mecanismo de protección que es la ofuscación. Se analiza una transformación de ofuscación del estado del arte y propone una mejora sobre la misma, elevando el nivel de protección y dificultando la tarea de los posibles atacantes a la propiedad intelectual.

Jara (2012), en su trabajo de investigación "Ofuscación de Permutaciones en MIXNETS", Universidad de Chile - Santiago de Chile; muestra que una ofuscación de esta naturaleza (MIXNETS) es posible de realizar, mostrando una construcción genérica de red de

mezcla que utiliza ofuscación para esconder el proceso de permutación y re-encriptación, dentro de cada servidor de mezcla. Además, a fin de mejorar la eficiencia de la construcción genérica, se propone como prueba de concepto implementable una construcción específica restringida a dos mensajes y un servidor de mezcla. Finalmente, para ambas construcciones se demuestra la seguridad de la ofuscación y la propiedad de anonimato en redes de mezcla.

b) Nacionales

León (2005), en su proyecto de investigación " Encriptación RSA de Archivos de Texto", Pontificia Universidad Católica del Perú – Lima-Perú; basa la seguridad en el hecho de que no existe una forma eficiente de factorizar números que sen productos de dos grandes primos. Las claves públicas y privadas se calculan a partir de un número que se obtiene como producto de dos primos. El RSA se basa en el uso de una función matemática que es fácil de calcular pero es difícil de invertir. La única manera de invertirla es utilizando la clave privada. Este tipo de funciones es conocida como funciones trampa de un solo sentido pues requieren que se conozca un secreto (que se conozca la trampa).

2.2. MARCO CONCEPTUAL

2.2.1. ALGORITMO

Según, Joyanes (2008), algoritmo es el conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problemas se denomina algoritmo, en otras palabras un algoritmo es un método para resolver un problema.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto. Los pasos para la resolución de un problema son:

a) Definición del problema, esta fase está dada por el enunciado del problema, en el cual requiere una definición clara y precisa.

b) Análisis del problema, una vez que ha se comprendido lo que se desea del computador, es necesario definir:

- Los datos de entrada.
- Cual es la información que se desea producir.
- Los métodos y fórmulas que se necesitan para procesar los datos.

c) Diseño del algoritmo, que describe la secuencia ordenada de pasos sin ambigüedades que conducen a la solución de un problema dado.

d) Codificación, es la operación de escribir la solución del

problema en una serie de instrucciones detalladas en un código reconocible por la computadora. Ejecución y validación del programa por la computadora.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que las ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en computadoras distintas. Las características fundamentales que debe cumplir todo algoritmo son:

- Precisión. Cada paso de un algoritmo debe estar precisamente definido; las operaciones a llevar a cabo deben ser especificadas de manera rigurosa y no ambigua para cada caso.
- Definido: si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Carácter finito. Un algoritmo siempre debe terminar después de un número finito de pasos.

La definición de un algoritmo debe describir tres partes:

- Entrada.
- Proceso.
- Salida.

2.2.2. DISEÑO DE ALGORITMOS

Según Valenzuela (2003), una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen, como ya se conoce el algoritmo.

La información proporcionada al algoritmo constituye su entrada y la información producida por el algoritmo se constituye su salida.

Métodos para el diseño de algoritmos

- **Recursión.**- La recursividad es una técnica fundamental en el diseño de algoritmos eficientes, que está basada en la solución de versiones más pequeñas del problema, para obtener la solución general del mismo. Una instancia del problema se soluciona según la solución de una o más instancias diferentes y más pequeñas que ella.

  Es una herramienta poderosa que sirve para resolver cierto tipo de problemas reduciendo la complejidad y ocultando los detalles del problema.

  Esta herramienta consiste en que una función o procedimiento se llama a sí mismo. Una gran cantidad de algoritmos pueden ser descritos con mayor claridad en términos de recursividad, típicamente el resultado será que sus programas serán más pequeños.

- **Divide y vencerás.-** Los problemas complejos se pueden resolver más eficazmente con la computadora cuando se rompen sub problemas que sean más fáciles de solucionar que el original. Consiste en dividir un problema más complejo en otros más simples.

- **Programación Dinámica(diseño ascendente)** .- La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples que pueden ser implementados para su solución en las computadoras .

  Puede ocurrir que la división natural del problema conduzca a un gran número de sub ejemplares idénticos. Si se resuelve cada uno de ellos sin tener en cuenta las posibles repeticiones, resulta un algoritmo ineficiente; en cambio si se resuelve cada ejemplar distinto una sola vez y se conserva el resultado, el algoritmo obtenido es mucho mejor.

  La programación dinámica es un método ascendente. Se resuelven primero los sub ejemplares más pequeños y por tanto más simples. Combinando las soluciones se obtienen las soluciones de ejemplares sucesivamente más grandes hasta llegar al ejemplar original.

- **Algoritmos Ávidos.-** Los algoritmos ávidos o voraces (Greedy Algorithms) son algoritmos que toman decisiones de corto alcance, basadas en información inmediatamente disponible, sin importar consecuencias futuras.

  Suelen ser bastante simples y se emplean sobre todo para resolver problemas de optimización, como por ejemplo, encontrar la secuencia óptima para procesar un conjunto de tareas por un computador, hallar el camino mínimo de un grafo, etc.

- **Método de Retroceso (backtracking).-** El Backtracking o vuelta atrás es una técnica de resolución general de problemas mediante una búsqueda sistemática de soluciones. El procedimiento general se basa en la descomposición del proceso de búsqueda en tareas parciales de tanteo de soluciones (trial and error).

  Las tareas parciales se plantean de forma recursiva al construir gradualmente las soluciones. Para resolver cada tarea, se descompone en varias subtareas y se comprueba si alguna de ellas conduce a la solución del problema. Las subtareas se prueban una a una, y si una subtarea no conduce a la solución se prueba con la siguiente.

La descripción natural del proceso se representa por un árbol de búsqueda en el que se muestra como cada tarea se ramifica en subtareas. El árbol de búsqueda suele crecer rápidamente por lo que se buscan herramientas eficientes para descartar algunas ramas de búsqueda produciéndose la poda del árbol.

2.2.3. ESTRATEGIAS DE PROGRAMACIÓN PARA LA IMPLANTACIÓN DE ALGORITMOS

Díaz (2003), manifiesta que si bien es cierto la implantación de los algoritmos diseñados puede efectuarse prácticamente en cualquier tipo de lenguaje de programación, no debemos pasar por alto que existen lenguajes de programación que funcionan de manera más eficiente bajo ciertas características de los algoritmos. Entonces dependiendo de las características de la solución diseñada usando algoritmos debemos elegir el paradigma de programación que pueda implementar de manera más simple dicho diseño.

- **Programación imperativa.-** También conocida como programación procedural, está basada en los lenguajes imperativos, que son la forma tradicional de programación y es totalmente antagónica con la programación en lenguajes declarativos.

Este tipo de programación se basa en la especificación explícita de los pasos que deben de seguir para obtener el resultado deseado, es decir, toma canónicamente la definición de algoritmo, en el sentido de una serie de pasos con lógica y secuencia para alcanzar un objeto determinado, implantándola en su estilo de programación. Utiliza variables y operaciones de asignación para el almacenamiento de información, define procedimientos bien claros en donde se procesan las variables de acuerdo con operaciones explícitas.

- **Programación lógica.-** Es un tipo de programación declarativa y relacional que está basada en lógica de primer orden. La programación declarativa describe relaciones entre variables en términos de funciones y reglas de inferencia, dejando en manos del traductor la aplicación de un algoritmo fijo sobre estas relaciones para producir un resultado. La programación de un algoritmo fijo sobre estas relaciones para producir un resultado. La programación relacional genera salidas en función de atributos y argumentos.

- **Programación funcional.-** Este paradigma basa su programación en la definición de un conjunto de funciones y una expresión cuyo valor de salida se utilizará para representar el resultado del algoritmo. La programación

funcional está basada en un tipo de lenguaje declarativo, ya que explicado en el punto anterior y en cálculo lambda tipificado con constantes. Las funciones de este tipo de programas no modifican su salida con entradas iguales, es decir, no tienen efectos laterales, lo que implica también que cumple con la regla de transparencia referencial. Los programas escritos en lenguaje funcional tienden a ser compactos y elegantes, aunque también son lentos y requieren de gran cantidad de memoria para su ejecución. Algunos de los principales lenguaje de programación funcional son Clean, FP, Haskell, Hope, LML, Miranda, SML,LISP.

- **Programación Orientada a Objetos.-** La programación Orientada a Objetos tiene por principio la representación de los objetos de la realizada en un lenguaje de programación que permita acercarse lo más posible al pensamiento humano, así como su modo de concebir y representar la realidad. A partir de este principio, la Programación Orientada a Objetos postula la identificación de los objetos que participen en el problema o situación que se quiere solucionar a través de un algoritmo. Realizada la identificación de objetos, se lleva a cabo una abstracción de sus características más relevantes, así como de las principales operaciones, que efectúan.

Posteriormente, se busca asociar objetos con características y operaciones comunes para llega a la definición de una clase.

Una clase posee atributos, interfaz y métodos. Los atributos son características o datos que tiene las clases, la interfaz es la parte de la clase que posibilita la comunicación entre objetos, los métodos son las operaciones o funciones asociadas a los objetos a través de los cuales se pueden modificar los valores que tienen los atributos de los objetos de la clase.

Un objeto es una instancia de una clase, esto implica que un objeto siempre tendrá una clase asociada de la cual hereda su estructura. La herencia debe entenderse como la extensión de las características de una clase hacia otra clase o hacia un objeto en particular.

La Programación Orientada a Objetos también postula el principio de ocultamiento de información que se basa en la definición de una parte pública y otra privada del objeto.

La parte pública puede ser accedida y modificada por los métodos públicos del objeto, mientras que la parte pública sólo puede accederse y modificarse usando los métodos

privados; estos últimos sólo pueden ser invocados desde los métodos públicos.

2.2.4. OFUSCACIÓN

Según Miralles(2005), la ofuscación del código, como técnica de seguridad, es actualmente una de las mejores herramientas de protección frente al "reverse engineering". Proporciona un software ininteligible pero con la misma funcionalidad que el código fuente original.

Según Jara(2012), Ofuscar es alterar el código de un programa para que sea ilegible; una definición de ofuscación más específica depende del contexto en que se aplique y las definiciones de seguridad que se usen, pero usualmente requieren que el programa ofuscado revele lo mínimo de información posible y más específicamente que ofusque todo tipo de claves secretas que se requieran para el funcionamiento del programa.

En informática la ofuscación es el acto deliberado de realizar un cambio no destructivo, ya sea en el código fuente de un programa informático o código máquina cuando el programa está en forma compilada o binaria, con el fin de que no sea fácil de entender o leer.

El código ofuscado es aquél código que, aunque se tiene el código fuente, ha sido enrevesado específicamente para ocultar su funcionalidad (hacerlo ininteligible).

La ofuscación binaria se realiza habitualmente para impedir o hacer más difícil los intentos de ingeniería inversa y desensamblado que tienen la intención de obtener una forma de código fuente cercana a la forma original.

Como un efecto lateral, la ofuscación, en ocasiones, hace que los programas resultantes sean más pequeños (aunque puede hacer que los programas sean más grandes en otros casos). Algunos tienden más a la ofuscación que otros. C, C++ y Perl son los más citados como fácilmente ofuscables. Las macros de preprocesador son usadas a menudo para crear código complicado de leer enmascarando la gramática y sintaxis estándar del lenguaje del cuerpo principal de código.

2.2.5. TÉCNICAS DE OFUSCACIÓN

Según Miralles (2005), un ofuscador es un programa que aplica transformaciones al código fuente de una aplicación en PHP. Esta transformación de ofuscación se clasifica en tres técnicas:

- Ofuscación de estructura,
- Ofuscación de control

- Ofuscación de datos.

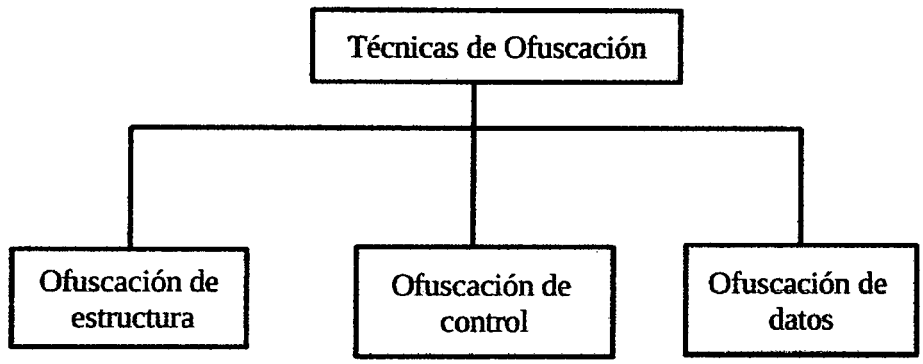


Figura 01: Técnicas de ofuscación

- **Ofuscación de estructura**

La ofuscación de estructura modifica la estructura del código mediante dos métodos básicos: Renombrar identificadores y borrar información redundante. Esto hace que el código contenga menor información con el objetivo de evitar la ingeniería inversa. La mayoría de las ofuscaciones de estructura no pueden deshacerse porque usan una función de un solo sentido que renombra identificadores mediante símbolos aleatorios y borra comentarios, métodos sin uso e información redundante. Pese a esto, la ofuscación de estructura no evita los ataques de ingeniería inversa ya que el código aún puede ser estudiado y comprendido. Este tipo de ofuscación es la más empleada y extendida.

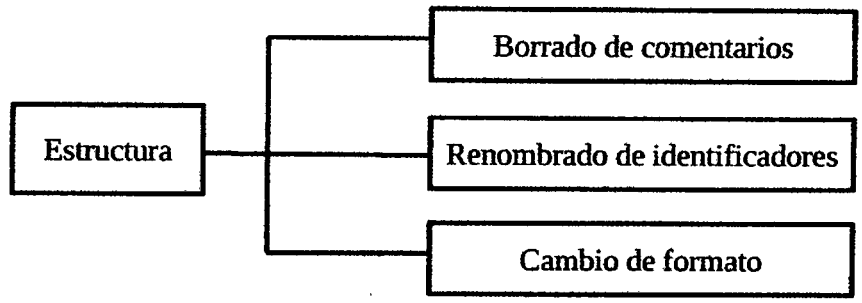


Figura 02: Ofuscación de estructura

- **Ofuscación de control**

  La ofuscación de control cambia el flujo de control del código fuente del programa. Esta técnica de ofuscación se clasificarla en:

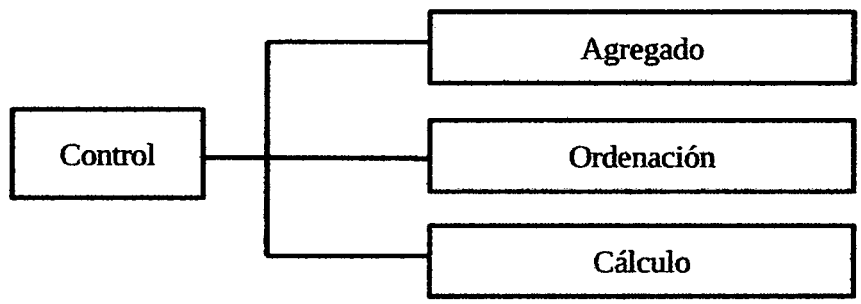


Figura 03: Ofuscación de control

- **Ofuscación de datos**

  Tiene por objetivo el oscurecimiento de datos y estructuras de datos que pueda haber en el código de la aplicación. La ofuscación de datos rompe las estructuras de datos usadas en el código y las encripta literalmente. Se tienen diferentes métodos para ofuscar las estructuras.

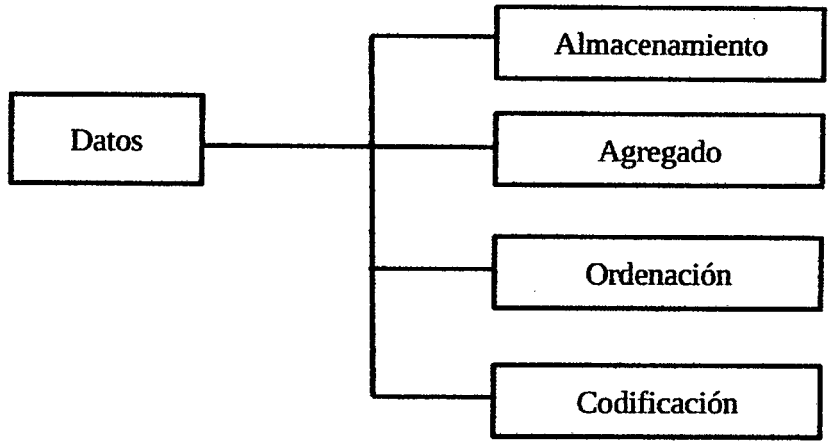


Figura 04: Ofuscación de datos

Según Dolz y Parra (2006), se entiende por ofuscar un código fuente o un código intermedio, un proceso mediante el cual se transforma mediante la aplicación de algoritmos de reescritura, un código perfectamente legible y entendible por una persona en otro de funcionalidad equivalente en un ciento por ciento, pero, en términos ideales, totalmente ilegible e incomprensible para un lector humano.

En pocas palabras, algunas de las técnicas de ofuscación más comunes consisten en la inclusión de bucles irrelevantes, cálculos innecesarios, comprobaciones fuera de contexto, nombres de funciones y de variables que no tienen nada que ver con su cometido, funciones que no sirven para nada, interacciones inverosímiles entre variables y funciones, etc. Otras técnicas, sin embargo, son mucho más potentes, en el sentido de que requieren un conocimiento superior de las características del lenguaje, e incluso pueden estar

diseñadas para burlar a herramientas de ingeniería inversa específicas.

2.2.6. PROPIEDAD INTELECTUAL

Según la Organización Mundial de la Propiedad Intelectual (2013), la propiedad intelectual es un derecho patrimonial de carácter exclusivo que otorga el Estado por un tiempo determinado para usar o explotar en forma industrial y comercial las invenciones o innovaciones, tales como un producto técnicamente nuevo, una mejora a una máquina o aparato, un diseño original para hacer más útil o atractivo un producto o un proceso de fabricación novedoso; también tiene que ver con la capacidad creativa de la mente: las invenciones, las obras literarias y artísticas, los símbolos, los nombres, las imágenes y privilegios.

El titular de la propiedad intelectual tiene la facultad para evitar que cualquier persona tenga acceso o haga uso de su propiedad sin su consentimiento.

Los derechos de propiedad intelectual que otorga cada país son independientes entre sí, por lo que una misma idea, invención, obra o carácter distintivo puede ser objeto de protección en una pluralidad de Estados, existiendo tantos títulos de protección como Estados que la hayan otorgado.

La propiedad intelectual se clasifica en dos categorías:

- **Propiedad industrial:** La propiedad industrial es el derecho exclusivo que otorga el Estado para usar o explotar en forma industrial y comercial las invenciones o innovaciones de aplicación industrial o indicaciones comerciales que realizan individuos o empresas para distinguir sus productos o servicios ante la clientela en el mercado. Esta incluye las invenciones, marcas, patentes, dibujos y modelos industriales, así como indicaciones geográficas de origen.

- **Derechos de autor:** La Ley de Derechos de Autor y el Decreto Supremo 061-62-DE regulan los derechos de autor en el Perú. Sin embargo la Constitución del país confiere a los tratados internacionales un rango mayor que la ley nacional.

Como se expresa en la citada ley los derechos de autor comprenden a todas las obras o producciones de carácter creativo, científico y artístico cualquiera que sea la forma de expresarlo.

Los derechos de autor protegen entre otros, los siguientes tipos de creaciones:

- **Obras literarias:** Artículos, libros, folletos, escritos de cualquier naturaleza; diccionarios enciclopedias, antologías, guías, y compilaciones de toda clase, así como lecciones, planes, discursos, sermones, memorias y obras de naturaleza similar, tanto en forma oral, como escrita o grabada. Se integran en este rango las versiones de manera íntegra o parcial de discursos expuestos en celebraciones de carácter oficial o científico, además de las publicaciones tales como: revistas o diarios, entre otros.

- **Obras artísticas:** Aquellas obras de carácter artísticos realizadas con o sin texto, obras teatrales en general; coreografías; composiciones musicales; obras producidas por radio o televisión y adaptaciones en estos medios de cualquier obra literaria; obras cinematográficas; dibujos, pinturas y similares; grabados, litografías, fotografías, entre otras formas de expresión artística.

- **Obras científicas:** Proyectos arquitectónicos, sistemas de creación de mapas, obras de carácter plástico y obras similares, así como cualquier otro arte o ciencia.

- **Otras obras:** Lemas, frases, títulos traducciones, adaptaciones de una obra con su correspondiente autorización, cualquier otra

obra o creación no incluida en las modalidades anteriores, como es el caso del software.

2.2.7. SOFTWARE

Pressman (2002), define al Software como un elemento del sistema que es lógico, comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware. Por tanto el software tiene unas características considerablemente distintas a las del hardware:

- El software se desarrolla, no se fabrica en un sentido clásico.
- El software no se estropea.
- Aunque la industria tiende a ensamblar componentes, la mayoría del software se construye a medida.

El software puede aplicarse en cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales, es decir un algoritmo (excepciones notables a esta regla son el software de los sistemas expertos y de redes neuronales).

El contenido y el determinismo de la información son factores importantes a considerar para determinar la naturaleza de una

aplicación de software. El contenido se refiere al significado y a la forma de la información de entrada y salida.

Algunas veces es difícil establecer categorías genéricas para las aplicaciones del software que sean significativas. Conforme aumenta la complejidad del software, es más difícil establecer compartimentos nítidamente separados. Las siguientes áreas del software indican la amplitud de las aplicaciones potenciales:

- **Software de sistemas**. Es un conjunto de programas que han sido escritos para servir a otros programas.
- **Software de tiempo real.**- El software que coordina, analiza y controla sucesos del mundo real conforme ocurren, se denomina de tiempo real.
- **Software de gestión.**- El proceso de la información comercial constituye la mayor de las áreas de aplicación del software.
- **Software de ingeniería y científico.**- El software de ingeniería y científico está caracterizado por los algoritmos de manejo de números.
- **Software empotrado.**- Los productos inteligentes se han convertido en algo común en casi todos los mercados de consumo e industrias.
- **Software de computadoras personales.**- El mercado del software de computadoras personales ha germinado en las

pasadas dos décadas.

- **Software basado en Web.-** Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, CGI, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales).
- **Software de inteligencia artificial.-** El software de inteligencia artificial (IA) hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo.

2.2.8. SOFTWARE LIBRE

Según Stallman M.(2004), software libre significa que el software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el software. Con estas libertades, los usuarios (tanto individualmente como en forma colectiva) controlan el programa y lo que hace.

Cuando los usuarios no controlan el programa, el programa controla a los usuarios. El programador controla el programa y, a través del programa, controla a los usuarios. Un programa que no es libre, llamado privativo, es por lo tanto un instrumento de poder injusto.

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa para cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que hagan los demás (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3).

Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones.

Un programa es software libre si los usuarios tienen todas esas libertades. Por tanto, debe ser libre de redistribuir copias, tanto con como sin modificaciones, ya sea gratuitamente o cobrando una tarifa por la distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir ni pagar el permiso.

También debe tener la libertad de hacer modificaciones y usarlas en privado para su propio trabajo o pasatiempo, sin siquiera mencionar que existen. Si publica sus cambios, no debe estar obligado a notificarlo a nadie en particular, ni de ninguna manera en particular.

La libertad de ejecutar el programa significa que cualquier tipo de persona u organización es libre de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y finalidad, sin que exista obligación alguna de comunicarlo al programador ni a específica. En esta libertad, lo que importa es el propósito de los usuarios, no el de los programadores. Como usuario es libre de ejecutar el programa para alcanzar sus propósitos, y si lo distribuye a otra persona, también esa persona será libre de ejecutarlo para lo que necesite.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las que no lo estén.

Software libre no significa que no es comercial. Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de software libre ya no es inusual; tal software libre comercial es muy importante. Puede haber pagado dinero para obtener copias de

software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el software, incluso de vender copias.

Si una modificación constituye o no una mejora, es un asunto subjetivo. Si su derecho a modificar un programa se limita, básicamente, a modificaciones que alguna otra persona considera una mejora, el programa no es libre. No obstante, eventuales reglas sobre cómo empaquetar una versión modificada son aceptables si no limitan substancialmente su libertad para publicar versiones modificadas, o su libertad para hacer y usar versiones modificadas en privado. Así, es aceptable que una licencia le obligue a cambiar el nombre de la versión modificada, eliminar el logotipo o identificar sus modificaciones como suyas.

Son aceptables siempre y cuando esas obligaciones no sean tan agobiantes que le dificulten la publicación de sus modificaciones. Como ya está realizando otras modificaciones al programa, no le supondrá un problema hacer algunas más.

2.2.9. PHP

Según Gutmans, Saether y Rethans(2005), el lenguaje de programación interpretado PHP nació como Personal Home Page

(PHP) Tools. Fue creado por el programador danés Rasmus Lerdorf en 1994 para la creación de páginas web dinámicas.

PHP (acrónimo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para desarrollo web y que puede ser incrustado en HTML.

El PHP suele utilizarse en interpretación del lado del servidor aunque también puede usarse desde una interfaz de línea de comandos y para la creación de otros tipos de programas.

Lerdorf diseñó la primera versión de PHP en lenguaje Perl con base en la escritura de un grupo de CGI del lenguaje C. Su intención era presentar su currículum vitae y almacenar datos como la cantidad de visitantes que accedían a su página web.

Los programadores israelíes Zeev Suraski y Andi Gutmans reescribieron el analizador sintáctico en 1997 y crearon el PHP3, cambiando el nombre del lenguaje al actual. Con el tiempo, estos

programadores reescribirían la totalidad del código de PHP.

Actualmente el PHP suele incrustarse dentro del código HTML de las páginas web y ejecutarse desde un servidor. Se estima que PHP está presente en más de veinte millones de sitios y en cerca de un millón de servidores.

Una de las ventajas de PHP es su parecido con lenguajes comunes de programación estructurada (como Perl y C), lo que ayuda a que los programadores puedan desarrollar aplicaciones complejas en poco tiempo.

2.2.10. CÓDIGO FUENTE

El código fuente de un programa informático (o software) es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

El código fuente de un programa está escrito por un programador en algún lenguaje de programación, pero en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a otro lenguaje (el lenguaje máquina o código objeto) que sí pueda ser ejecutado por el hardware de la computadora. Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes y otros sistemas de traducción.

El término código fuente también se usa para hacer referencia al código fuente de otros elementos del software, como por ejemplo el código fuente de una página web que está escrito en el lenguaje de marcado HTML o en Javascript u otros lenguajes de programación web y que es posteriormente ejecutado por el navegador web para visualizar dicha página cuando es visitada.

2.2.11. IMPLEMENTACIÓN DE ALGORITMO

Según Raffo (1999), en programación, los algoritmos se implementan en forma de sentencias en algún lenguaje de programación. De esta manera, la forma de escribir los algoritmos depende del lenguaje de programación, y del paradigma usado. Estos son los algoritmos que pueden ser interpretados por una computadora y así ser ejecutados.

Los algoritmos también pueden representarse gráficamente empleando diagramas de flujo o formas similares. De esta manera, son fácilmente comprensibles, especialmente para personas que no son programadores. También, de esta manera, los algoritmos son más "universales", pues no dependen de un lenguaje de programación específico. Los algoritmos también pueden escribirse en pseudocódigo, lo que también los hace fáciles de entender.

Se hacen intentos para que las computadoras interpreten y ejecuten los diagramas de flujo y los pseudocódigos, pero no logran la flexibilidad, potencia y velocidad de los algoritmos puramente escritos en un lenguaje de programación específico.

Un algoritmo también puede expresarse en lenguaje natural, aunque esto puede traer ambigüedades e interpretaciones erróneas (la ambigüedad es propia del lenguaje humano).

2.2.12. PROTECCIÓN DE SOFTWARE

Según Hernadez y Manzanares(2002), para proteger el software éste debe ser legal y estar legalmente constituido. Para ello se debe tener tanto un entorno operativo legal con su licencia adecuada.

Normalmente las herramientas de desarrollo vienen en varias versiones, cada una de ellas con unas características añadidas a la versión inmediatamente inferior. Generalmente de cada herramienta se suelen encontrar por lo menos las versiones "Personal", "Profesional" y "Enterprise" en que orden creciente de complejidad y coste. En la la mayoría de los casos, a los programas desarrollados con la versión "Personal" no se les puede dar un uso comercial, necesitando para ello las licencias "profesional" o "Enterprice".

2.2.13. NIVEL DE SATISFACCIÓN

Según Thomson, La satisfacción del cliente es el nivel del estado de ánimo de una persona que resulta de comparar el rendimiento percibido de un producto o servicio con sus expectativas.

2.3. MARCO CONCEPTUAL

2.3.1. DESARROLLADOR DE SOFTWARE EN PHP DE LA Región APURÍMAC

Desarrollador de software de la Región Apurímac que se dedica a la programación de código fuente PHP basado en la tecnología LAMP, esta persona puede contribuir a la visión general del proyecto más a nivel de aplicación que a nivel de componentes o en las tareas de programación individuales.

2.3.2. CREACIÓN DE SOFTWARE COMO PROPIEDAD INTELECTUAL

El derecho de autor es aplicable a cualquier software, con independencia de su originalidad y/o valoración económica. En su comercialización debe cuidarse que tanto el propio software y sus manuales como cualquier material publicitario lleven incorporadas las correspondientes inscripciones de "copyright" ya que es importante que el autor/propietario reclame para sí los derechos sobre su obra.

2.3.3. OFUSCAMIENTO DEL CÓDIGO PHP

El ofuscamiento PHP es una opción para los desarrolladores ya que el compilador permite la ejecución del código PHP mediante la función EVAL(), esto hace posible que el código sea transformado a un formato ininteligible para las terceras personas sin embargo este formato es interpretado por el compilador PHP sin perder la funcionalidad del código fuente original.

CAPÍTULO III

3. HIPÓTESIS Y VARIABLES

3.1. FORMULACIÓN DE HIPÓTESIS

a) HIPÓTESIS GENERAL

El algoritmo de ofuscación protege el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012.

b) HIPÓTESIS ESPECÍFICAS

- La creación del algoritmo de ofuscación "protege" el código fuente PHP como propiedad intelectual de los desarrolladores de software de la Región Apurímac 2012.

- El nivel de satisfacción de los desarrolladores de software es "aceptable" al usar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

3.2. SISTEMA DE VARIABLES

La investigación pretende proteger el código fuente basado en PHP como propiedad intelectual de los desarrolladores de software con la utilización del algoritmo de ofuscación de manera que se define las siguientes variables:

Variable independiente: Algoritmo de ofuscación.

Variable dependiente: Protección de código fuente PHP.

Variable interviniente: Nivel de satisfacción.

3.3. OPERACIONALIZACIÓN DE VARIABLES

VARIABLES	DIMENSIÓN	INDICADORES	ÍNDICES
INDEPENDIENTE Algoritmo de ofuscación	Análisis del algoritmo	• Tiempo de Ejecución • Complejidad	1.- Número de operaciones. 2.- Orden de complejidad.
DEPENDIENTE Protección de código fuente PHP	Codificación	• Ofuscamiento	1.- Alto. 2.- Mediano. 3.- Bajo.
INTERVINIENTE Nivel de satisfacción	Satisfacción de los desarrolladores	• Nivel de satisfacción	1.- Totalmente de acuerdo. 2. - De acuerdo. 3. - Ni de acuerdo, ni en desacuerdo. 4.- En desacuerdo. 5.- Totalmente de acuerdo

CAPÍTULO IV

4. DISEÑO METODOLÓGICO

4.1. TIPO Y NIVEL DE INVESTIGACIÓN

Tipo: Investigación aplicada de tipo pre experimental, ya que de acuerdo al problema es cubrir el vacío que tienen los desarrolladores de software que requieren proteger el código fuente PHP como propiedad intelectual, dado que la aplicación sea exitoso o sea fracaso.

Nivel: Descriptivo, por que pretende proteger el código fuente PHP como propiedad intelectual y analizar el nivel de satisfacción de los desarrolladores de software es aceptable en proteger el código fuente PHP como propiedad intelectual.

4.2. MÉTODO Y DISEÑO DE INVESTIGACIÓN

Se utilizará el método inductivo y deductivo, observando a los desarrolladores de software que se basan en PHP, con diseño pre experimental con dos poblaciones independientes para estudio de un solo caso.

Diseño:

P1 ----------> x ----------> R1

P2 ----------> x ----------> R2

Donde:

P1: Es protección de código fuente PHP.

P2: Es nivel de satisfacción de los desarrolladores de software en PHP.

R1: Resultado de la observación de la protección de código fuente PHP.

R2: Resultado de la observación del nivel de satisfacción.

4.3. MATERIAL DE INVESTIGACIÓN

Para la investigación se utilizó las herramientas de desarrollo como editores de PHP, un servidor donde estará la tecnología LAMP, dos computadoras de desarrollo con conexión a Internet.

4.4. EQUIPOS PARA LA INVESTIGACIÓN

Los materiales utilizados para la investigación son:

- Herramientas de desarrollo como editores de PHP.
- Servidor con la tecnología LAMP.
- Computadoras cliente para desarrollo con conexión a Internet.

4.5. INSTRUMENTOS Y RECOLECCIÓN DE DATOS

La técnica que se ha usado es el registro de datos y cuestionario para el nivel de satisfacción de los desarrolladores en PHP.

Los instrumentos a utilizados son:

- Registro de datos al utilizar el algoritmo ofuscación probar si protege el código fuente PHP.
- Cuestionario aplicado a los desarrolladores de software con respecto al test de nivel de satisfacción.

4.6. PLAN DE TRATAMIENTO DE DATOS

4.6.1. PRUEBA DE HIPÓTESIS PARA EL ALGORITMO DE OFUSCACIÓN

a) **Formulación de hipótesis**

Ho: El algoritmo de ofuscación no protege el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

H1: El algoritmo de ofuscación protege el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

b) **Nivel de significancia**

$\alpha=0.05$

c) **Estadístico**

Como las muestras son pequeñas son pequeñas se ha utilizado la distribución binomial.

$$P(X=16)=\frac{n!}{X!(n-X)!}.p^{X}.(1-p)^{n-X}$$

Donde:

P(X)= 16 ---------> Probabilidad de X éxitos dadas n y p

n= 20 ---------> Número de observaciones

p=0,5 ---------> Probabilidad de éxito

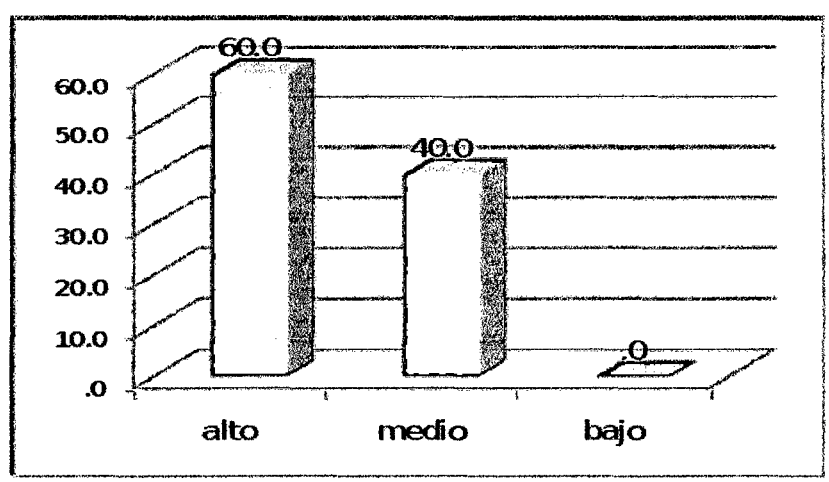
1-p= 0,5 ---------> Probabilidad de fracasos

X= 16 ---------> Número de éxitos en la muestra (X=0,1,2,3,4,5,.....n)

Tabla 01: Calificación sobre el grado de ofuscación

	n	%
alto	16	60.0
medio	4	40.0
bajo	0	.0
Total	20	100.0

Figura 05: Calificación sobre el algoritmo de ofuscación



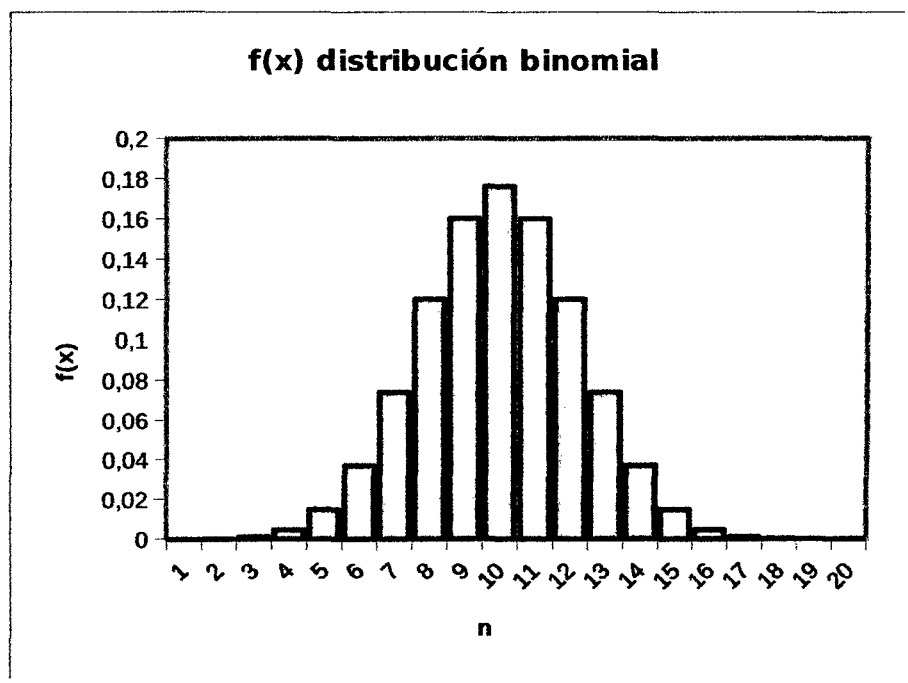
Como se observa en la tabla 01 y figura 05, el 60 % de los programadores consultados le dio un calificativo de alto seguido por el 40% con un calificativo de regular, no existiendo calificativo de bajo para este programa.

Calculando:

$$P(X=16)=\frac{n!}{X!(n-X)!}.p^{X}.(1-p)^{n-X}$$

$$P(X=16)=\frac{20!}{16!(20-16)!}.0{,}5^{16}.(1-0{,}5)^{20-16}$$

$$P(X=16)=0{,}00462$$



Decisión: Como la prueba calculada 0.00462 es mayor a la tabla, entonces se rechaza a la H0 y se acepta a la hipótesis H1, esto indica

que al algoritmo de ofuscación protege el código fuente PHP como propiedad intelectual de los desarrolladores de la Región Apurímac.

4.6.2. PRUEBA DE HIPÓTESIS PARA EL NIVEL DE SATISFACCIÓN

a) **Formulación de hipótesis**

Ho: El nivel de satisfacción de los desarrolladores de software es "aceptable" al usar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual en la Región Apurímac 2012..

H1: El nivel de satisfacción de los desarrolladores de software no es "aceptable" al usar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual en la Región Apurímac 2012.

b) **Nivel de significancia**

$\alpha=0.05$

c) **Estadístico**

Como las muestras son pequeñas son pequeñas se ha utilizado la distribución chi-cuadrado.

$$x^2=\frac{\sum (n1-n2)^2}{n2}$$

La tabla muestra las frecuencias observadas y las frecuencias esperadas al aplicar la encuesta para la satisfacción del usuario.

Tabla 02: Calificación sobre el nivel de satisfacción

Frecuencia	Muy satisfecho	Satisfecho	Ni satisfecho, ni insatisfecho	Insatisfecho	No sirve
Esperada	3	2	0	0	0
Total Observada	5	5	5	5	5

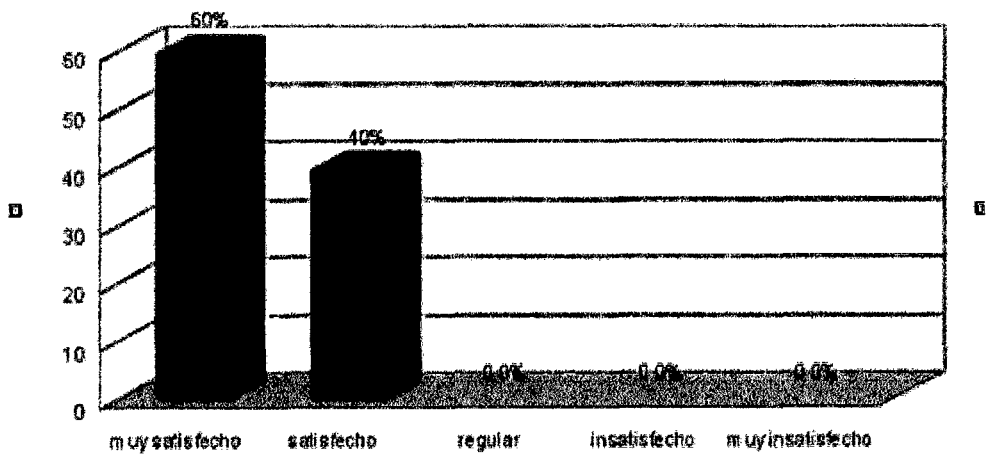


Figura 06: Pruebas de hipótesis nivel de satisfacción

Como se observa en la figura 06 en relación con la tabla 02, el 60 % de los programadores consultados le dio un calificativo de alto seguido por el 40% con un calificativo de regular, no existiendo calificativo de bajo para este programa.

Dónde:

r = 2 Número de clasificación del problema.

k= 5 Número de parámetros estimados.

gl= (r-1)(k-1)=4 Grados de libertad.

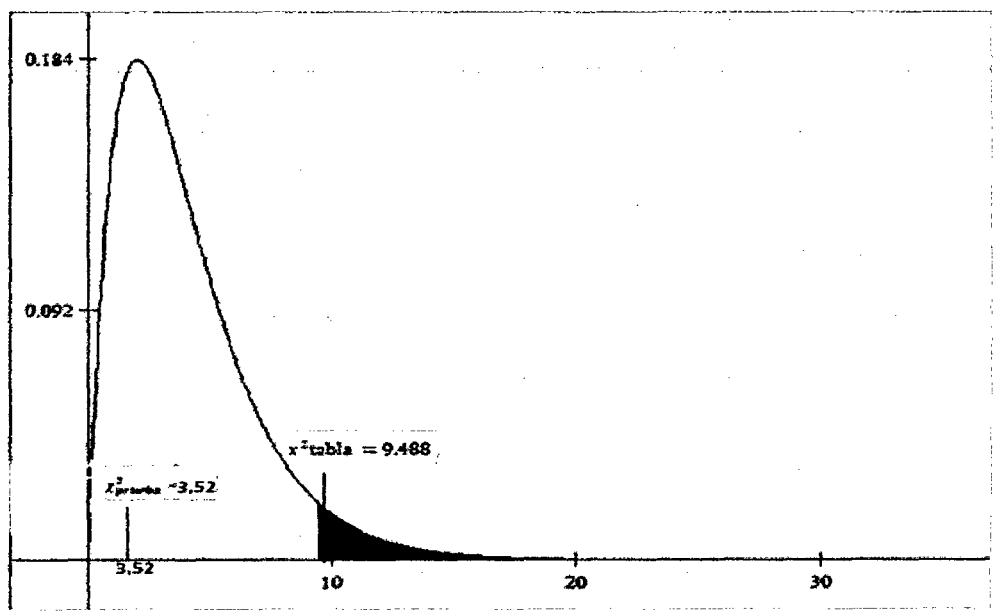
Con lectura en la tabla con 4 grados de libertad y 0,05 de área se obtiene: x^2tabla =9,488

Calculando:

$$X^2_{prueba}=\sum \frac{(O_i+e_i)^2}{e_i}$$

$$X^2_{prueba}=\sum \frac{(3-5)^2}{5}+\frac{(2-5)^2}{5}+\frac{(0-5)^2}{5}+\frac{(0-5)^2}{5}+\frac{(0-5)^2}{5}$$

$$X^2_{prueba}=3,52$$



Decisión: Ho es aceptada, ya que $_{X^2 prueba}$ = 3,52 es menor que x^2tabla =9,488, por lo tanto, se concluye que el nivel de satisfacción de los desarrolladores es aceptable al utilizar el algoritmo de ofuscación de código fuente PHP.

Tabla 03: Varianza total explicada del nivel de satisfacción

Componente	Autovalores iniciales			Sumas de las saturaciones al cuadrado de la extracción			Sumas de las saturaciones al cuadrado de la rotación	
	Total	% de la varianza	% acumulado	Total	% de la varianza	% acumulado	Total	% de la varianza
1	4.854	60.678	60.678	4.854	60.678	60.678	3.849	48.116
2	1.628	20.347	81.024	1.628	20.347	81.024	2.056	25.705
3	1.250	15.625	96.649	1.250	15.625	96.649	1.826	22.829
4	.268	3.351	100.000					
5	.000	.000	100.000					
6	.000	.000	100.000					
7	.000	.000	100.000					
8	.000	.000	100.000					

Tabla 04: Varianza total explicada del nivel de satisfacción

Matriz de componentes rotados				
		Componente		
		1	2	3
p1	¿Qué le parece el manejo del algoritmo de ofuscación?	.180	-.089	.979
p2	¿Cuenta con información acerca de su uso?	.958	.240	-.153
p3	¿Recibe múltiples archivos y carpetas como entrada?	.205	.866	-.384
p4	¿Aplicaría el algoritmo para proteger su código fuente como propiedad intelectual?	.948	.182	.260
p5	¿Recomendaría a otros utilizar el algoritmo?	.948	.182	.260
p6	¿Publicaría en los servidores los archivos ofuscados como sus sistema final?	.347	.618	.555
p7	¿Cómo calificaría el algoritmo de ofuscación?	.948	.182	.260
p8	¿Cuál es su nivel de satisfacción?	.196	.871	.227

En la tabla Nro 03, se realizó un análisis factorial de que tanto es posible explicar los calificativos obtenidos en la tabla 04. En donde se encontró el 60% de los resultados son posibles explicar por la pregunta (pa2) del cuestionario aplicado a los programadores se

observa la carpeta de salida de los archivos ofuscados, ¿Cuenta con información acerca de su uso? y por (pa4) ¿Aplicaría el algoritmo para proteger su código fuente como propiedad intelectual?. En tanto que el 20.34% es explicado por las preguntas: (pa5) ¿Recomendaría a otros utilizar el algoritmo?, (pa7) ¿Cómo calificaría el algoritmo de ofuscación?.

CAPÍTULO V

5. RESULTADOS Y DISCUSIÓN

5.1. ESTRATEGIA PARA EL DISEÑO DEL ALGORITMO

Diseño de algoritmos

Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutarse por la máquina constituyen, como ya se conoce el algoritmo.

Para el diseño del algoritmo se ha utilizado el método de la programación dinámica, ya que se ha descompuesto problema original en subproblemas más simples así como:

- Subida del archivo local al servidor.
- Descompresión del archivo subido.
- Validación de archivos.
- Ofuscación.

- Compresión.
- Descarga.

Estos subproblemas hacen que sea más fácil el desarrollo del algoritmo, además cada uno tiene una función encargada de realizar las tareas asignadas.

5.2. ESTRATEGIA PARA LA PROGRAMACIÓN DEL ALGORITMO

El desarrollo de algoritmos se puede hacer en cualquier lenguaje de programación, sin embargo existen lenguajes de programación en los que el algoritmo funciona más eficientemente bajo ciertas características del algoritmo. Para este proyecto se ha usado el lenguaje PHP, ya que éste cumple con los paradigmas de programación que hacen más simple el diseño.

En esta investigación además de utilizar el lenguaje PHP, se tuvo en cuenta el paradigma de la programación imperativa, ya que permite al desarrollador escribir código que describe detalladamente los pasos que se debe realizar para cumplir el objetivo, por eso es importante definir con cuidado la entrada a cada función y qué devuelve cada función.

- Para convertir caracter a OCTADECIMAL se utiliza la función:

```
function foct($char){
        return decoct(ord($char));
        }
```

- Para conseguir la extensión de un dirección de archivo.

```
function GetEXT($fpath){
        $ext = explode('.',$fpath);
        $ext = strtolower($ext[count($ext)-1]);
        return $ext;
}
```

5.3. CREACIÓN DEL ALGORITMO

Luego de hacer un análisis acerca de la protección intelectual, se ha creado el algoritmo para la ofuscación de código fuente PHP para la protección intelectual utilizando la técnica de ofuscación de control. Los objetivos incluían tanto investigar métodos, maneras de obtener un algoritmo capaz de ofuscar el código fuente PHP, y proteger la propiedad intelectual.

ALGORITMO PRINCIPAL

1.- Inicio

2.- subir archivo.ext

3.- si (errordesubida > 0)

        escribir "error de subida"

        Salir

4.- ext <— extensión del archivo subido

  //modo de ofuscación simple u optimizado

5.- leer modo

6.- si (ext = php)

        // obtiene id a base del tiempo del servidor

        ftmp <— idtiempo_nombredearchivo.php

```
            // abrir archivo en modo escritura
            fh <— abrir archivo (ftmp,w)
            fcont <— leer todo el contenido del archivosubido.php
            fcont <— Ofuscar(fcont,modo)
            // escribe en el archivo fh la cadena ofuscada
            escribir archivo (fh,fcont)
            escribir "descargar archivo ofuscado"
            Salir
 // si la extensión es : .txt, doc, xls, jpg, png, gif, ...etc
    si (ext < > zip)
                escribir "el archivo no es .zip "
                Salir
    //instancia una clase para leer el objeto zip subido
7.- za <— nuevo archivo_coprimido()
    //instancia una clase para escribir el objeto zip ofuscado
8.- zip <— nuevo archivo_comprimido()
    // obtiene id a base del tiempo del servidor
9.- fzip <— idtiempo_nombredearchivo.zip
     //es la ruta y el nombre del archivo.zip resultado
10.- filename <— /tmp/fzip
       //abre el archivo en modo escritura para grabar los archivos   ofuscados
11.- si (zip <— abrir archivo(filename,w) < > true)
          escribir "no se puede crear zip resultado"
          Salir
12.- si (za <— abrir archivo(archivo_subido.zip < >  true)
          escribir "no se puede abrir el archivo subido "
           Salir
```

```
        // vacía toda la información sobre en za
       vaciar información (za)
      //recorriendo archivos del zip subido
13.- para i<— 0  hasta i < (za <— num_archivos)
          si (archivo[i] < > carpeta)
                    fcont <— za <—leer(archivo[i])
                    //extensión del archivo
                    ext <— extensión archivo[i]
               si (ext = php)
                    //Ofuscar PHP
                    fcont <— Ofuscar(fcont,modo)
               sino
                    si(ext =js)
                         //Ofuscar JS
                         fcont <— MinJS(fcont)
                         si(ext = css)
                              //Ofuscar CSs
                              fcont <— MinCSS(fcont);
                              //agregando el archivo a zip
                              zip —> agregar(archivo[i],fcont);
 14.- escribir "Descargar offs_nombrearchivo.zip"
15.- Salir
16.- Fin proceso
```

FUNCIÓN GENERAL OFUSCAR

```
función Ofuscar(contenido,modo)
1.- INICIO
2.- contenido <— comprimir(contenido)
    // hace una llamada a la función MinHTML para minimizar el código html
3.- contenido <— MinHTML(contenido)
    //número de ofuscaciones random en el rango de 1 a 40
4.-  r <— rand(1,40)
5.- para i <— 0 hasta i<r
          //llamada a la función OfPHP para ofuscar con HEX y OCT
          contenido <— OfPHP(contenido)
6.- si (modo=verdad)
     //llamada a la función OfPHPs para ofuscar en modo simple
7.- contenido <— OfPHPs(contenido)
8.- retornar contenido
FIN
```

FUNCIÓN OFUSCAR MODO SIMPLE

```
funcion OfPHPs(str)
1.- INICIO
    // Devuelve el valor ASCII  específico de un carácter con una id en un
       random  de a,z
2.- ideco <— devolver caracter(random(a,z)+idtiempo)
    //Devuelve el valor ASCII  específico de un carácter con una id en un
      random  de a,z
3.- igzin <— devolver caracter(random(a,z)+idtiempo)
```

```
   //descomprimiendo una cadena en octadecimal y hexadecimal
4.- strd <— igzin <— fstr('descomprimir cadena')
5.- strd <— evaluar(igzin(ideco(decodifica64(comprimir(str)))))?>"
6.- retornar strd;
7.- FIN
```

FUNCIÓN OFUSCAR MODO OPTIMIZADO ESTADO DE ARTE

```
//función ofuscar en modo optimizado
funcion OfPHP(contenido)
1.- INICIO
2.- minstr <— contenido;
3.- minstr <— buscar comentarios y reemplazar con vacio("/'.*?'/","",minstr);
   //cuenta inicio de php <?
4.- n1 <— substr_count(minstr,'<?')
   //cuenta el fin de php ?>
5.- n2 <— substr_count(minstr,'?>');
6.- si(n1>n2)
        //agrega fin de php ?>
        minstr <— minstr+"?>"
7.- contenido <— decodifica64(comprimir(contenido))
8.- tam <— tamaño(contenido)
   //define el número de partes que se dividirá la cadena
9.- npartes <— random(menor que tam)
10.- partes <— array()
11.- ptam <— redondear(tam/npartes)
12.- vars <— array()
13.- para i<—0 hasta i*ptam<=tam
```

```
    //Devuelve el valor ASCII  específico de un carácter con una id en un random de a,z
        uid <— devuelve caracter(random(a,z)+idtiempo)
        //crea el valor para esa variable
        partes[uid] <— sustraer(contenido,i*ptam,ptam)
        //almacena el orden de las variables
        vars[] <— uid;
    //pone en secuencia las variables concatenándolos.
14.- final <— secuencia de variables (vars)
    //desordena el orden de las variables
15.- desordenar orden variable(partes)
16.- para i<—0 hasta partes
        //lista en el nuevo orden las variables con sus contenidos
        stringData <— stringData+partes[i]+random(0,1)+fstr(partes[i])+enter
    //prefijo de variable
17.- ideco <— random(a,z)+idtiempo
18.- stringData <— stringData+"ideco +fstr('decodifica 64')
    //Devuelve el valor ASCII  específico de un carácter con una id en un random de a,z
19.- igzin <— devuelve caracter(random(a,z)+idtiempo)
    //sufijo + descomprime la cadena comprimida en hexadecimal y octadecimal
20.- stringData <— stringData + igzin + fstr('descomprimir cadena')
    // función final eval
21.- stringData <— evalua(igzin(ideco(final)))
22.- return "<?php\n"+stringData+"\n?>"
23.- FIN
```

```
// función convertir cadena a Hexadecimal y Octadecimal
funcion fstr(str)
1.- Inicio
2.- tstr <— " ";
3.-para(i<— 0;i<strlen(str);i++)tstr.= (rand(0,1)?"\\'.foct(str[i]):'\\x'.fhex(str[i]));
5. -retornar tstr;
Fin Proceso
```

5.4. PRUEBA DE ESCRITORIO DEL ALGORITMO DE OFUSCACIÓN

contenido <— <?php echo "HOLA"; /* comentario*/
funcion OfPHP(contenido)

No	SENTENCIA	RESULTADO
1	**INICIO**	
2	minstr <— contenido;	<?php echo "HOLA"; /* comentario*/
3	minstr <— buscar comentarios y reemplazar con v	<?php echo "HOLA";
4	n1 <— substr_count($minstr,'<?')	1
5	n2 <— substr_count($minstr,'?>')	0
6	si(n1>n2)	VERDAD
	minstr <— minstr+"?>"	<?php echo "HOLA"; ?>
7	contenido <— codifica64(comprimir(contenido))	s7EvyChQSE3OyFdQ8vD3cVSyVrC3AwA=
8	tam <— tamaño(contenido)	32
9	npartes <— random(menor que tam)	4
10	partes <— array()	partes es un array
11	ptam <— redondear(tam/npartes)	8
12	vars <— array()	vars es un array
13	**i=0**	**i*ptam <= tam**
	i = 0	0*8 <= 32 verdad
		uid=51ed9af456eb3
		partes[51ed9af456eb3]=subtraer(s7EvyChQSE3OyFdQ8vD3cVSyVrC3AwA=,0,8) = **s7EvyChQ**
		var[]=51ed9af456eb3
	i = 1	1*8 <= 32 verdad
		uid=53ed9zf456eb2
		partes[53ed9zf456eb2]=subtraer(s7EvyChQSE3OyFdQ8vD3cVSyVrC3AwA=,8,8) = **SE3OyFdQ**
		var[]=53ed9zf456eb2
	i = 2	2*8 <= 32 verdad
		uid=98ed9zf456eb7
		partes[98ed9zf456eb7]=subtraer(s7EvyChQSE3OyFdQ8vD3cVSyVrC3AwA=,16,8) = **8vD3cVSy**
		var[]=98ed9zf456eb7
	i = 3	3*8 <= 32 verdad
		uid=98ed9zf456eb7
		partes[98ed9zf456eb7]=subtraer(s7EvyChQSE3OyFdQ8vD3cVSyVrC3AwA=,24,8) = **VrC3AwA=**

		var[]=98ed9zf456eb7
	i = 4	4*8 <= 32
14	final <- secuencia de variables (vars)	secuencia de variables
15	desordenar orden variable(partes)	desordena en el mismo array
16	para i<—0 hasta npartes	**stringData <- stringData+partes[i]+random(0,1)+fstr(partes[i])+enter**
	i = 0	51ed9af456eb3 = "\66\66\x52\70\x39\x44\165\102" +
	i = 1	53ed9zf456eb2= "\124\x4b\X39\X79\x6c\67\x6a\132\" +
	i = 2	98ed9zf456eb7 ="\x59\124\x7a\142\x37\x41\165\164" +
	i = 3	98ed9zf456eb7 ="\63\x66\x58\70\x48\105\x55\170"
17	ideco <— random(a,z)+idtiempo	**b51c4c454cef8e**
18	stringData <— stringData+"ideco +fstr('decodifica 64')	51ed9af456eb3 = "\66\66\x52\70\x39\x44\165\102" 53ed9zf456eb2= "\124\x4b\X39\X79\x6c\67\x6a\132\" 98ed9zf456eb7 ="\x59\124\x7a\142\x37\x41\165\164" 98ed9zf456eb7 ="\63\x66\x58\70\x48\105\x55\170" **b51c4c454cef8e=\147\172\151\x6e\x66\154\141\164\x65**
19	igzin<—devuelve caracter(random(a,z) +idtiempo)	**y51c4c454ce127**
20	stringData <— stringData + igzin + fstr('descomprimir cadena')	51ed9af456eb3 = "\66\66\x52\70\x39\x44\165\102" 53ed9zf456eb2= "\124\x4b\X39\X79\x6c\67\x6a\132\" 98ed9zf456eb7 ="\x59\124\x7a\142\x37\x41\165\164" 98ed9zf456eb7 ="\63\x66\x58\70\x48\105\x55\170" **b51c4c454cef8e=\147\172\151\x6e\x66\154\141\164\x65 Y51c4c454ce127 = \142\141\163\x65\x36\x34\137\144\145**
21	stringData <— evalua(igzin(ideco(final)))	51ed9af456eb3 = "\66\66\x52\70\x39\x44\165\102" 53ed9zf456eb2= "\124\x4b\X39\X79\x6c\67\x6a\132\" 98ed9zf456eb7 ="\x59\124\x7a\142\x37\x41\165\164" 98ed9zf456eb7 ="\63\x66\x58\70\x48\105\x55\170" **b51c4c454cef8e=\147\172\151\x6e\x66\154\141\164\x65 Y51c4c454ce127 = \142\141\163\x65\x36\x34\137\144\145** Eval(51ed9af456eb3 + 53ed9zf456eb2 + 98ed9zf456eb7 + 98ed9zf456eb7)
22	return "<?php\n"+stringData+"\n?>"	return "<?php\n 51ed9af456eb3 = "\66\66\x52\70\x39\x44\165\102" 53ed9zf456eb2= "\124\x4b\X39\X79\x6c\67\x6a\132\" 98ed9zf456eb7 ="\x59\124\x7a\142\x37\x41\165\164" 98ed9zf456eb7 ="\63\x66\x58\70\x48\105\x55\170" **b51c4c454cef8e=\147\172\151\x6e\x66\154\141\164\x65 Y51c4c454ce127 = \142\141\163\x65\x36\x34\137\144\145** Eval(51ed9af456eb3 + 53ed9zf456eb2 + 98ed9zf456eb7 + 98ed9zf456eb7) \n?>"
23	FIN	

5.5. ANÁLISIS DE LA EFICIENCIA DE ALGORITMOS

función Ofuscar(contenido,modo)

1. INICIO 1OE

2. contenido <— comprimir(contenido) 2OE

3. contenido <— MinHTML(contenido) 2OE

4. r <— rand(1,40) 2OE

5. para i<—0 hasta i<r

   contenido <— OfPHP(contenido) .....$n(n^2+17n+C)$

6. si (modo=verdad) 2OE

7. contenido <— OfPHPs(contenido) .....2OE+n+C

8. retornar contenido 1OE

9. FIN

$T(n) = n^3+17n^2+n+C$

función OfPHPs(str)

1. INICIO 1OE

2. ideco <— devolver caracter(random(a,z)+idtiempo) 3OE

3. igzin <— devolver caracter(random(a,z)+idtiempo) 3OE

4. strd <— igzin<—fstr('descomprimir cadena') ..... 3OE+n+C

5. 5.- strd <— evaluar(igzin(ideco(decodifica64(comprimir(str)))))?>"..5OE

6. retorna strd; 1OE

8. FIN 1OE

$T(n) = n+C$

function OfPHP(content)

1. Inicio 1OE

2.- minstr <— contenido; 1OE

3.-minstr <— buscar comentarios y reemplazar con vacío("/.*?/","",minstr);
.....2OE

4.- n1 <— substr_count(minstr,'<?') 2OE

5.- n2 <— substr_count(minstr,'?>'); 2OE

6.- si(n1>n2) 2OE

7.- contenido <— decodifica64(comprimir(contenido)).....3OE

8.- tam <— tamaño(contenido) 2OE

9.- npartes <— random(menor que tam) 3OE

10.- partes <— array() 2OE

11.- ptam <— redondear(tam/npartes) 3OE

12.- vars <— array() 2OE

13.- para i<—0 hasta i*ptam<=tam

uid <— devuelve caracter(random(a,z)+idtiempo) 3n

partes[uid] <— sustraer(contenido,i*ptam,ptam) 5n

vars[] <— uid; 1n

14.- final <— secuencia de variables (vars) 2OE

15.- desordenar orden variable(partes) 1OE

16.- para i<—0 hasta partes

stringData<—stringData+partes[i]+random(0,1)+fstr(partes[i]) enter

.....$6n+n^2+C$

17.- ideco <— random(a,z)+idtiempo 4OE

18.- stringData <— stringData+"ideco<—fstr('decodifica 64')....4OE+n+C

19.- igzin <— devuelve caracter(random(a,z)+idtiempo) 3OE

20.- stringData<—stringData+igzin<— fstr('descomprimir cadena') .....4OE+n+C

21.- stringData <— evalua(igzin(ideco(final))) 4OE

22.- return "<?php\n"+stringData+"\n?>" 1OE

23.- FIN 1OE

$T(n) = n^2+17n+C$

funcion fstr(str)

1.- Inicio 1OE

2.- tstr <— " "; 1OE

3.-para(i<—0;i<strlen(str);i++tstr.=(rand(0,1)?'\\'.foct(str[i]):'\\x'.fhex(str[i]));

.....n

5. -retornar tstr; 1OE

6.- FIN

$T(n) = n+C$

Donde T(n) = Tiempo empleado para ejecutar el algoritmo con una entrada de tamaño n.

Por lo cual , tomando el mayor valor, en el peor caso se concluye que: El tiempo de ejecución del algoritmo es de :

$T(n)=Max(n^3+17n^2+n+C, n+C, n^2+17n+C) = n^3+17n^2+n+C$

Y la complejidad del algoritmo es orden polinomial decir $O(n^3)$.

5.6. DESCRIPCIÓN DE LA IMPLEMENTACIÓN DEL ALGORITMO

El objeto de la creación del algoritmo de ofuscación es la ofuscación del código fuente PHP, para la protección intelectual de los desarrolladores de software en PHP de la Región Apurímac.

Esquema del proceso y funcionamiento del algoritmo de Ofuscación

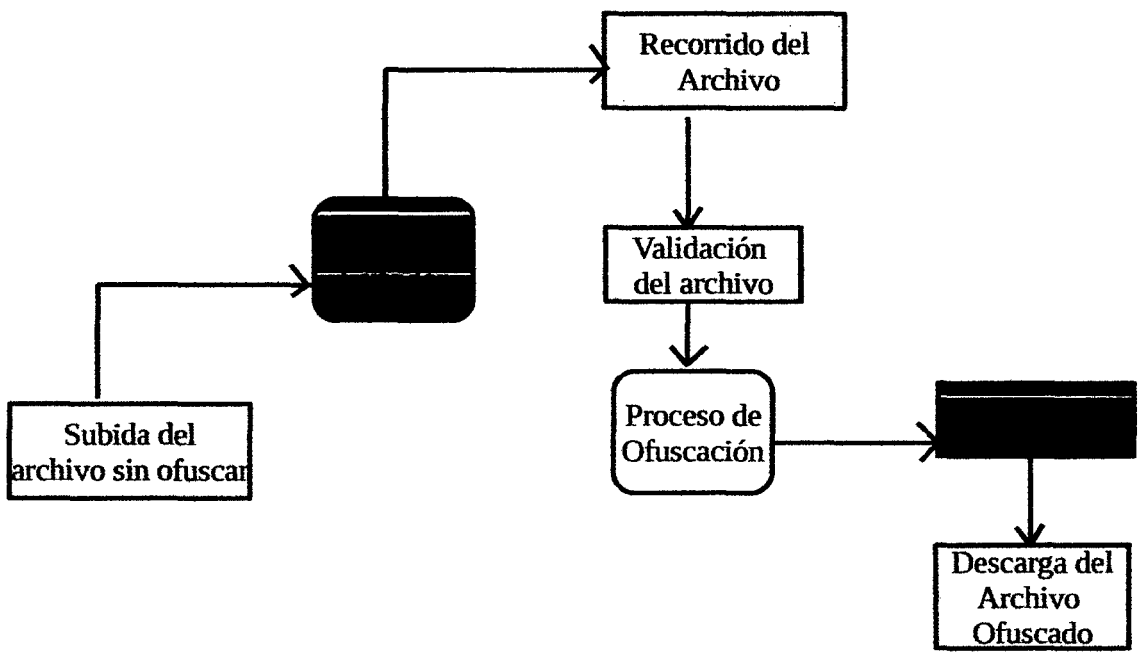


Figura 07: Esquema del proceso de ofuscación
Fuente. Elaboración propia

Descripción de los módulos

- **Subida del archivo local al servidor.-** El servidor peticiona un software comprimido en zip, y hecho en php.
- **Descompresión del archivo subido.-** Una vez que el archivo está en el servidor, se descomprime el archivo.
- **Validación de archivos.-** Se verifica que esté en PHP.

- **Ofuscación.-** Se ofusca los archivos en PHP, los demás archivos que no estén en PHP quedan como estaban.
- **Compresión.-** Una vez ofuscado el software se comprime.
- **Descarga.-** En ésta se procede a la descarga en donde peticiona donde o en que carpeta ha de ser descargado.

Entrada de datos

El algoritmo recibe como entrada mediante un formulario archivos independientes .php o un sistema en su conjunto en formato .zip. Seguidamente valida los archivos proporcionados.

Procesamiento

Ofusca cada archivos .php, dejando sin modificaciones el resto de los archivos tales como: .jpg .swf .js etc, seguidamente obtiene un número aleatorio que sirve para hacer la cantidad de divisiones del código ofuscado con la finalidad de desordenar, creando un cierto número de variables asignados bloques de divisiones en hexadecimales y octadecimales, finalmente construye un cadena para la ejecución del la función eval() de PHP.

Salida de datos

Los archivos php ofuscados son transferidos y empaquetados .zip agregando a esto la copia exacta del resto de los archivos como: .jpg, .js .swf etc, finalmente el sistema prevee de descarga del sistema ofuscado .zip.

5.7. ETAPAS DE LA IMPLEMENTACIÓN DEL ALGORITMO DE OFUSCACIÓN DE CÓDIGO FUENTE PHP.

ETAPA I.- Diseño de la interfaz de la aplicación del algoritmo, tiene la función de lanzar al público el algoritmo, en la cual pide el software a ser ofuscado, si quiere la ofuscación en modo simple.

ETAPA II: Validación de los datos de entrada:

Exclusión de archivos .php en la ofuscación, ya que alguno de estos archivos al estar ofuscado no permite la modificación que hay veces es necesario, por ejm., el archivo config.php por su constante modificación de los datos de acceso a la base de datos no es recomendable ofuscarlo.

```
$noinc = array("config.php","tcpdf/","excel/");
```

Verificación de los archivos de la subida de los archivos para la entrada del algoritmo:

```
if ($_FILES["file"]["error"] > 0) die("Error: " . $_FILES["file"]["error"]
. "<br>".$retorna);
```

Si hay error de subida de archivos entonces mandará un mensaje de error.

En caso de que el archivo subido es independiente .php, se validará para el ofuscamiento directamente llamando a la función Ofuscar($fcont, $modosimple).

```
if($ext=='php'){
        $ftmp = uniqid()."_".$_FILES["file"]["name"];
        $fh = fopen("./tmp/".$ftmp, 'w') or die("no se puede crear archivo");
        $fcont = file_get_contents($_FILES["file"]["tmp_name"]);
        $fcont = Ofuscar($fcont,$modosimple);
        fwrite($fh, $fcont);
        fclose($fh);
```

Si el archivo de entrada es un software empaqueta en .zip, entonces se procederá primero a crear un objeto para descomprimir un archivo subido asignándole un archivo temporal .zip para el resultado, recorre todos los archivos del .zip verificando que cumplan con las condiciones establecidas. Una vez verificado procede hace una llamada a la función ofuscar.

```
$za  = new ZipArchive();
$zip = new ZipArchive();
$fzip = uniqid()."_".$_FILES["file"]["name"];
$filename = "./tmp/".$fzip;
if($zip->open($filename, ZIPARCHIVE::CREATE)!==TRUE) die("no
se puede crear archivo zip resultado".$retorna);
if($za->open($_FILES["file"]["tmp_name"])!==TRUE) die("no se puede
abrir archivo zip subido".$retorna);
var_dump($za);
```

```
for ($i=0; $i<$za->numFiles;$i++) {
        $if = $za->statIndex($i);
        $fpath = $if['name'];
        echo "index: $i - name: ".$fpath."\n";
            if(!preg_match("/\/$/",$fpath)){
                $fcont = $za->getFromName($fpath);
                $ext = GetEXT($fpath);
                if($ext == 'php'&&NoArray($noinc,$fpath) ){
                    $fcont = Ofuscar($fcont,$modosimple);
                }else if($ext == 'js'){
                    $fcont = MinJS($fcont);
                    }else if($ext == 'css'){
                        $fcont = MinCSS($fcont);
                            }
                $zip->addFromString($fpath, $fcont);
                        }
                    }
```

ETAPA III.- Ejecución del algoritmo:

Una vez validado los datos de entrada el sistema llama a la función Ofuscar($content,$mds) que directamente ejecuta el algoritmo de ejecución sea simple u optimizado, indicando el número de iteraciones que va a realizar al ofuscar el software.

```
function Ofuscar($content,$mds){
        $content = compress_php_src($content);
        $content = MinHTML($content);
        $r = rand(1,3);
        for($i=0;$i<$r;$i++) $content = OfPHP($content);
        if($mds==true)$content = OfPHPs($content);
        return $content;
}
```

ETAPA IV.- Compresión y empaquetamiento del sistema ofuscado, utilizando la función: compress_php_src($src).

Una vez ofuscado los archivos .php, se crea un archivo .zip donde son transferidos todo los .php ofuscados, agregando el resto de los archivos .jpg, .js .swf etc, seguidamente el sistema permite hacer un download del sistema completo mediante un formulario.

ETAPA V.- Una vez ofuscado el software, dará como resultado un archivo empaquetado .zip listo para descargarlo y ponerlo en funcionamiento.

5.8. PASOS PARA OFUSCAR UN SOFTWARE

- Se tiene establecer una conexión con el servidor, para ello, se conecta A la red o Internet del servidor, si es en la red local la invocación ser hará desde una PC configurada con IP de la misma red.

- Una ver logrado la conexión, desde el navegador web se hace la llamada al servidor donde esta almacenado el algoritmo, como se muestra en la siguiente imagen.

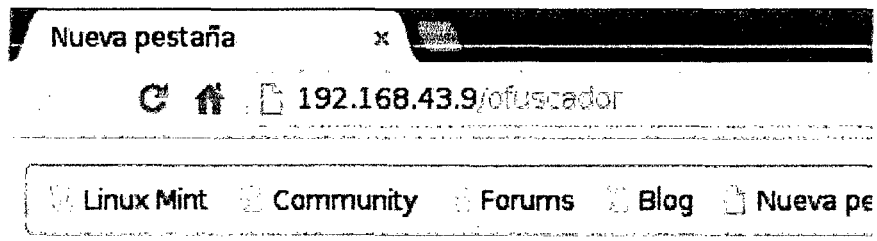


Figura 08: Conexión con el algoritmo de ofuscación.

- Una vez ejecutado, nos muestra la siguiente imagen, en la cual indica la selección del archivo a ofuscar y que debe estar comprimido en .zip y debe ser de la extensión php.

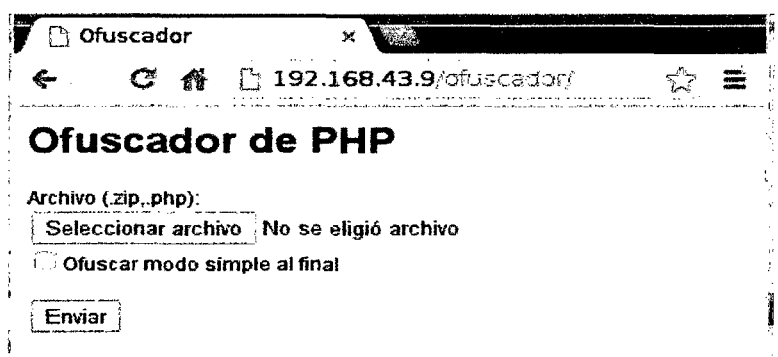


Figura 09: Selección del algoritmo para la ofuscación.

- Seleccionamos el archivo comprimido en .zip, luego hacemos clic en enviar teniendo como resultado lo siguiente.

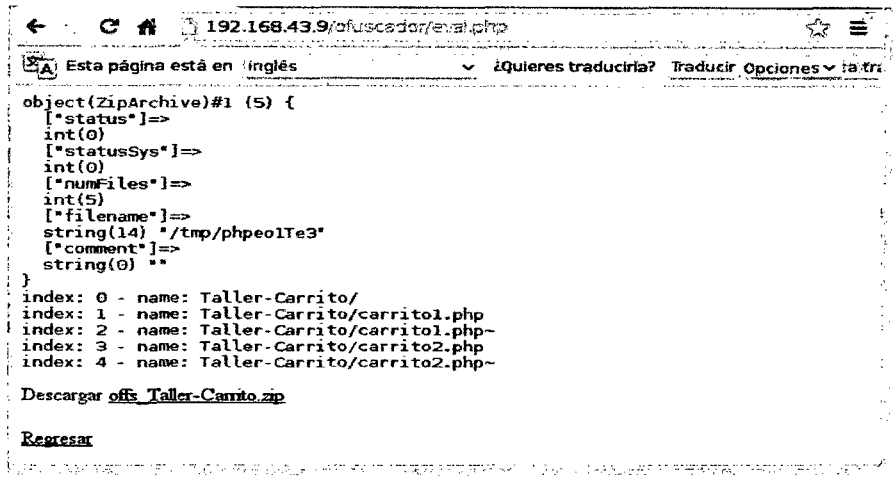


Figura 10: Ventana de salida del algoritmo ofuscado

- Descargamos el archivo ofuscado.
- Descomprimimos el archivo ofuscado, y se observa que el archivo se descomprime en su nombre original como se muestra en la siguiente figura.
- Abriendo la carpeta del software ofuscado se observa que los archivos no han cambiado su nombre y que el contenido ha sido ofuscado.
- Ejecutamos el software ofuscado obteniendo la misma funcionalidad del software.

5.9. ANÁLISIS DEL SOFTWARE SEGÚN EL NÚMERO DE CARACTERES

- **Análisis del algoritmo algoritmo de ofuscación de código fuente PHP en modo simple.**

Tabla 05: Algoritmo de Ofuscación con simple 1 iteración

Número de caracteres por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB Al Ofuscar el Software
72189	77	25	-53
102718	145	68	-77
712703	1126	305	-821
2965601	5222	1331	-3891

En la tabla 05 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -53 Kb; al ofuscar un software con 102718 caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -77 Kb; al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -821 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -3891 Kb.

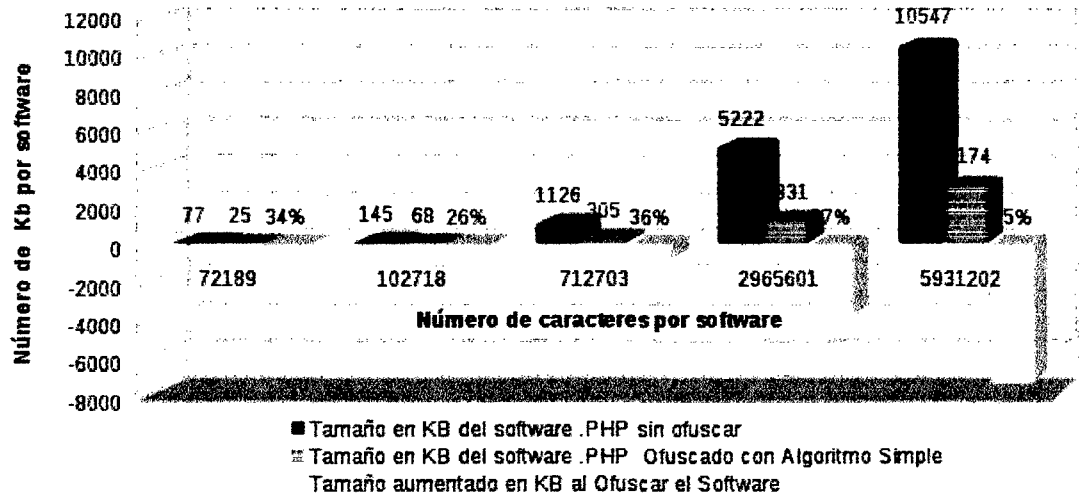


Figura 11: Resultados de la ofuscación del software en modo simple por número de caracteres con 1 iteración

Como se puede observar en el figura 11, el algoritmo de ofuscación de código fuente PHP de modo simple con 1 iteración, se reduce el tamaño en un promedio de 33,6 % KB.

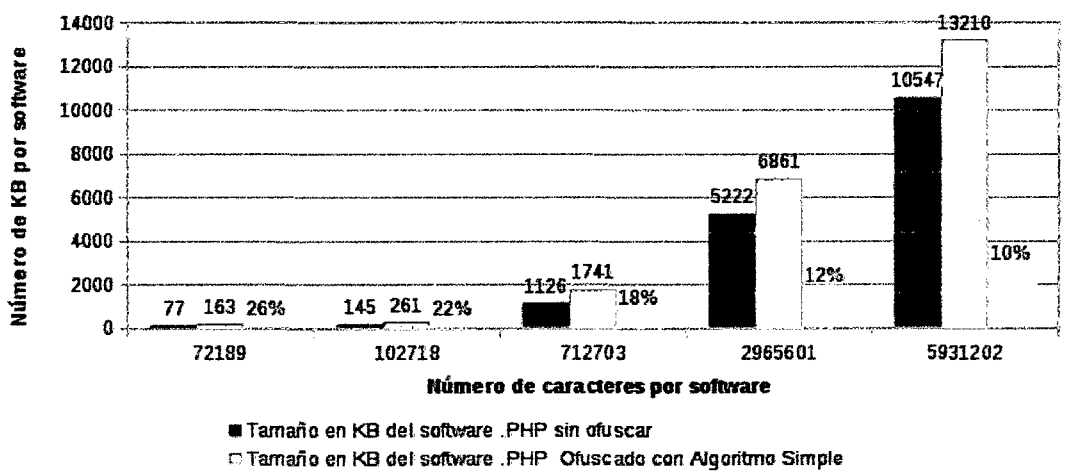
Tabla 06: Algoritmo de Ofuscación con simple 5 iteraciones

Número de caracteres por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB Al Ofuscar el Software
72189	77	163	86
102718	145	261	116
712703	1126	1741	614
2965601	5222	6861	1638
5931202	10547	13210	2662

En la tabla 6 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones aumenta el tamaño en 86 Kb; al ofuscar un software con 102718 caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones aumenta el tamaño a 116 Kb;

al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones, aumenta el tamaño a 614 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones, aumenta el tamaño a 1638kb; al ofuscar un software con 5931202 caracteres, 10547Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y cinco una iteraciones, aumenta el tamaño a 2662kb.

Figura 12: Resultados de la ofuscación del software en modo simple por número de caracteres con 5 iteraciones



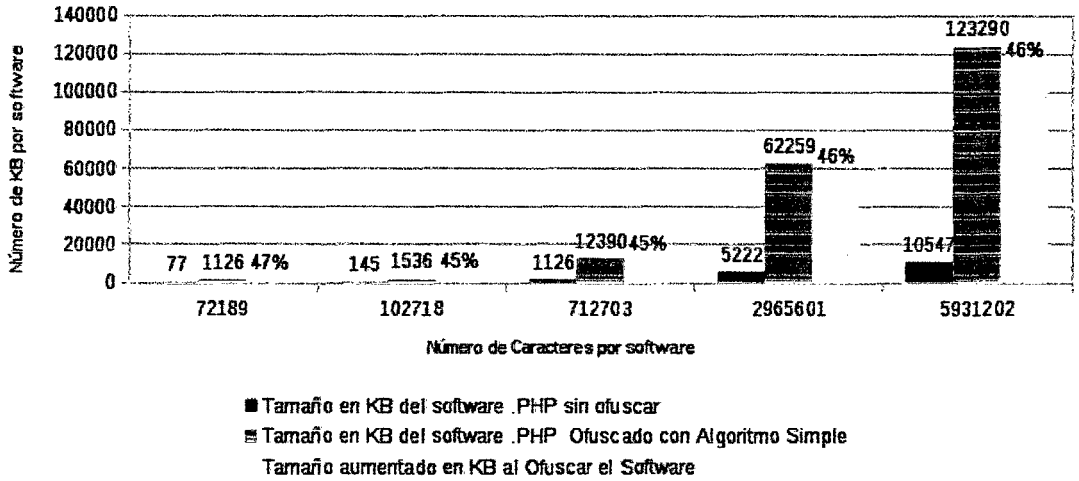
La figura 12 muestra que al ofuscar el software con algoritmo simple y con 5 iteraciones, el tamaño se incrementa en promedio de 17,6 % KB de los 5 softwares que se han tomado como muestra.

Tabla 07: Algoritmo de Ofuscación con simple 10 iteraciones

Número de caracteres Por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB Al Ofuscar el Software
72189	77	1126	1049
102718	145	1536	1391
712703	1126	12390	11264
2965601	5222	62259	57037
5931202	10547	123290	112742

En la tabla 07 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones aumenta el tamaño en 1049 Kb; al ofuscar un software con 102718 caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones aumenta el tamaño a 1391 Kb; al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 11264 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 57037kb; al ofuscar un software con 5931202 caracteres, 10547Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 112742kb.

Figura 13: Resultados de la ofuscación del software en modo simple por número de caracteres con 10 iteraciones



La figura 13 muestra que al ofuscar el software con algoritmo simple y con 10 iteraciones, el tamaño se incrementa en promedio de 45,8 % KB de los 5 softwares que se han tomado como muestra.

- **Análisis del algoritmo de ofuscación de código fuente php en modo optimizado.**

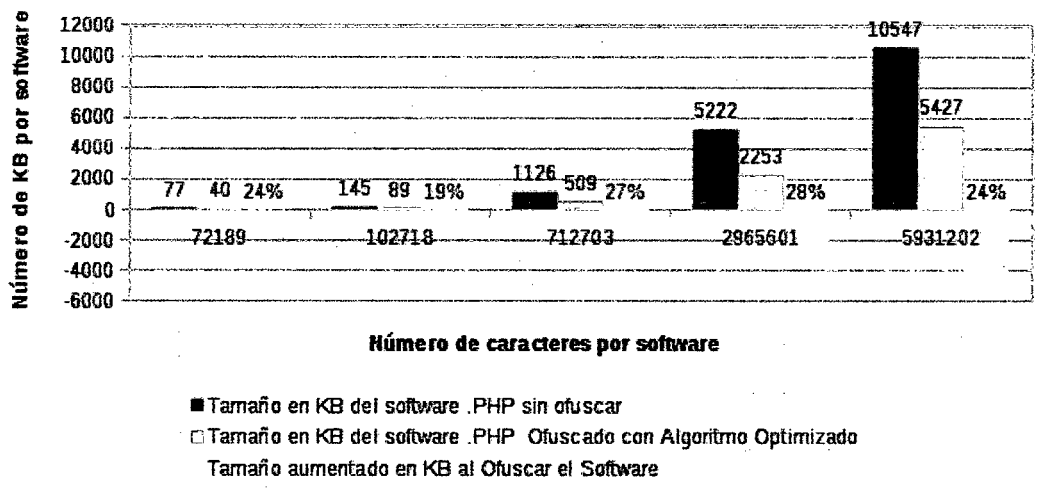
Tabla 08:Algoritmo de Ofuscación Optimizado con 1 iteración

Número de caracteres Por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB al Ofuscar el Software
72189	77	40	-38
102718	145	89	-56
712703	1126	509	-618
2965601	5222	2253	-2970
5931202	10547	5427	-5120

En la tabla 08 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -38 Kb; al ofuscar un software con 102718

caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -56 Kb; al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -618 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con una iteración disminuye a -2970 Kb.

Figura 14: Resultados de la ofuscación del software en modo optimizado por número de caracteres con 1 iteración.



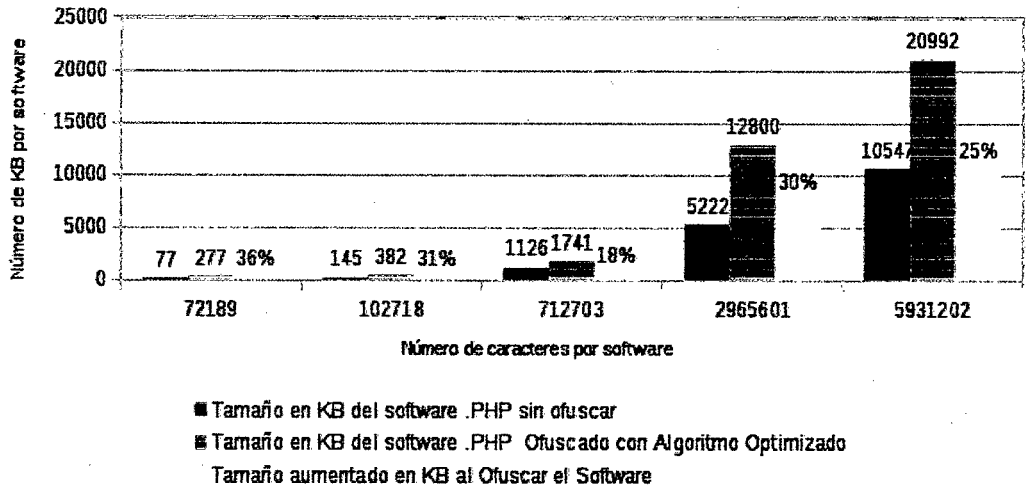
La figura 14 muestra que al ofuscar el software con algoritmo optimizado y con 1 iteración, se reduce el tamaño en promedio de 24,4 % KB de los 5 softwares que se han tomado como muestra.

Tabla 09: Algoritmo de Ofuscación Optimizado con 5 iteraciones

Número de caracteres Por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB al Ofuscar el Software
72189	77	277	200
102718	145	382	238
712703	1126	1741	614
2965601	5222	12800	7578
5931202	10547	20992	10445

En la tabla 09 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones aumenta el tamaño en 200 Kb; al ofuscar un software con 102718 caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones aumenta el tamaño a 238 Kb; al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones, aumenta el tamaño a 614 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con cinco iteraciones, aumenta el tamaño a 7578 Kb; al ofuscar un software con 5931202 caracteres, 10547Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y cinco una iteraciones, aumenta el tamaño a 10445 Kb.

Figura 15: Resultados de la ofuscación del software en modo optimizado por número de caracteres con 5 iteraciones.



La figura 15 muestra que al ofuscar el software con algoritmo optimizado y con 5 vueltas, se incremente el tamaño en promedio de 28 % KB de los 5 softwares que se han tomado como muestra.

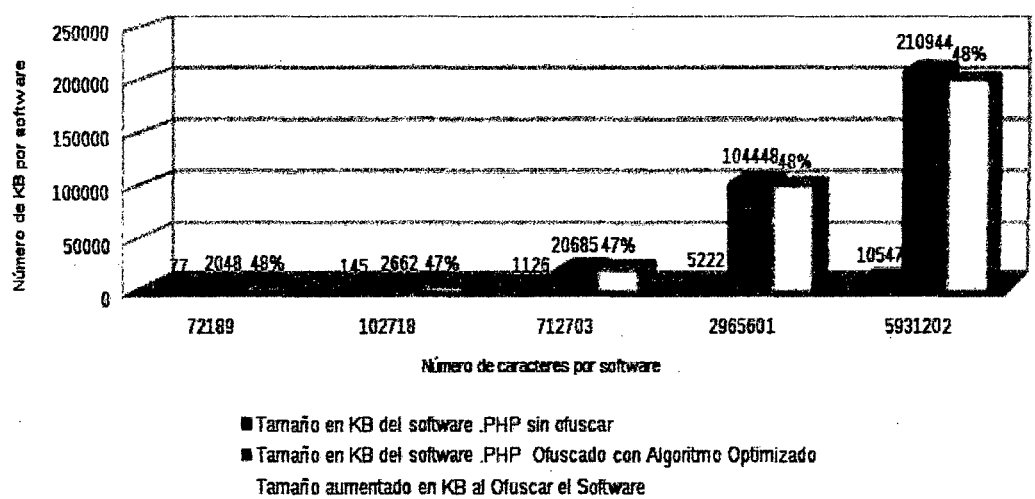
Tabla 10: Algoritmo de Ofuscación Optimizado con 10 iteraciones

Número de caracteres Por software	Tamaño en KB del software .PHP sin ofuscar	Tamaño en KB del software .PHP Ofuscado con Algoritmo Simple	Tamaño aumentado en KB al Ofuscar el Software
72189	77	2048	1971
102718	145	2662	2518
712703	1126	20685	19558
2965601	5222	104448	99226
5931202	10547	210944	200397

En la tabla 10 de observa que al ofuscar un software con 72189 caracteres, 77Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y

con diez iteraciones aumenta el tamaño en 1971 Kb; al ofuscar un software con 102718 caracteres, 145Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones aumenta el tamaño a 2518 Kb; al ofuscar un software con 712703 caracteres, 1126Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 19558 Kb; al ofuscar un software con 2965601 caracteres, 5222Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 99226 Kb; al ofuscar un software con 5931202 caracteres, 10547Kb, el tamaño en Kb del software PHP ofuscado con algoritmo simple y con diez iteraciones, aumenta el tamaño a 200397 Kb.

Figura 16: Resultados de la ofuscación del software en modo optimizado por número de caracteres con 10 iteraciones.



La figura 16 muestra que al ofuscar el software con algoritmo optimizado con 10 iteraciones, se incremente el tamaño en promedio de 47,6 % KB de los 5 softwares que se han tomado como muestra.

Como se ha observado el los cuadros de la ofuscación del software por número de caracteres y el modo de algoritmo a utilizar, se puede decir que el algoritmo puede realizar la ofuscación desde un solo archivo o software con un pequeño número de caracteres hasta grandes cantidades de las mismas.

De acuerdo al tipo de algoritmo que se utiliza sea simple o modo optimizado se va incrementado el tamaño en KB.

Cuanto más número de vueltas que se le aplique al algoritmo, el software será más ofuscado y difícil de entender, sin embrago el tamaño en KB va a seguir aumentado por cada iteración de la ofuscación.

En esta investigación se experimenta la creación del algoritmo de ofuscación de códigos fuente PHP para la protección intelectual de los desarrolladores de software en PHP de la Región Apurímac, como resultado los desarrolladores pueden exponer a terceros su código fuente bajo las condiciones que viere por conveniente. Por otro lado corroborando con el estudio " Ofuscadores de Código Intermedio" realizado por Dolz y Parra (2006), analizan que para la pérdida de la propiedad intelectual se puede utilizar un mecanismo de protección mediante la ofuscación. Se analiza una transformación de ofuscación del estado del arte y se propone una mejora sobre la misma, elevando el nivel de protección y dificultando la tarea de los posibles atacantes a la propiedad intelectual.

Según León (2005), en su proyecto de investigación utiliza la criptografía, en donde menciona que la protección de la información se lleva a cabo variando su forma. Se llama cifrado (o transformación criptográfica) a una transformación del texto original (llamado también texto inicial o texto claro) que lo convierte en el llamado texto cifrado o criptograma. Análogamente, se llama descifrado a la transformación que permite recuperar el texto original a partir del texto cifrado. La teoría de la información estudia el proceso de la transmisión ruidosa de un mensaje: Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos primos. El RSA se basa en el uso de una función matemática que es fácil de calcular pero es difícil de invertir. La única manera de invertirla es utilizando la clave privada. En cambio en esta investigación se realiza la ofuscación del código fuente PHP diferenciándose de la criptografía que usa calve pública y privada, la ofuscación desde técnica es confundir la comprensión del código fuente viéndose de esta manera ilegible.

CONCLUSIONES

Primera: Se ha creado el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual de los desarrolladores de software en la Región Apurímac 2012, cuyo funcionamiento está implantado en la plataforma LAMP en donde al aplicar el algoritmo a un código PHP perfectamente legible y entendible por el desarrollador se transforma en otro de funcionalidad equivalente, pero totalmente ininteligible e incomprensible para un lector humano.

Segunda: Se ha realizado pruebas de ofuscación para ver si el algoritmo de ofuscación protege o no el software, en donde según a los resultados de la prueba estadística binomial se obtuvo 0.00462 lo cual es mayor a la tabla, afirmando entonces que el algoritmo de ofuscación protege el código fuente PHP, para esto se hizo una comparación de los dos modos de ofuscación, el simple y el optimizado, logrando así que el código fuente del software que ha pasado por este proceso de ofuscación es totalmente ininteligible por terceros.

Tercera: Se ha determinado que el nivel de satisfacción de los desarrolladores de software en lenguaje de programación PHP de la Región Apurímac es aceptable en un 60% al utilizar el algoritmo de ofuscación para proteger el código fuente PHP como propiedad intelectual.

RECOMENDACIONES

Primero: El algoritmo de ofuscación motivo de esta investigación aún es posible mejorar su optimización sobre todo en la ocupación de espacio tratando en el número de iteraciones de ofuscaciones.

Segundo: Se ha intentado ejecutar la instrucción eval de un código ofuscado mediante el bash ofuscado realizado por Francisco Javier Rosales García de la Universidad Politécnica de Madrid, de manera que esta combinación optimiza más aún la ofuscación para la protección intelectual ocultando el eval de PHP, sin embargo solo funciona para archivos independientes PHP, por lo cual se sugiere investigar para que esta combinación funcione para un sistema en conjunto.

REFERENCIAS BIBLIOGRÁFICAS

- Alonso L. y Muñoz, (2009). "Propiedad intelectual - El uso de la marca como herramienta de mercado." Primera edición.
- Dolz, D. Y Parra, G.(2006). "Ofuscadores de Código Intermedio. Reporte Preliminar." Buenos Aires-Argentina. Universidad Nacional de Comahue.
- Gutmans,A.,Saether, Stig y Rethans, D.(2005). "PHP5 Power Programming".Indianápolis.
- Jara Rodríguez, M.(2012) ."Ofuscación de Permutaciones en MIXNET ".Santiago de Chile. Universidad de Chile.
- Joyanes Aguilar L.(2008). "Fundamentos de Programación". Desde McGraw-Hill Cuarta Edición.
- León Lomparte, K.(2005)."Encriptación RSA de Archivos de Texto". Lima-Perú. Pontificia Universidad Católica del Perú.
- Miralles Arévalo, A.(2005). Estudio y Prueba de una Métrica para los Ofuscadores de Java. Sevilla. Proyecto de Fin de Carrera. Universidad de Sevilla: Departamento de Sistemas y Automática.
- Pressman, R.(2005), "Ingeniería del Software". Sexta Edición.

- Raffo Lecca Eduardo (1999) "Algoritmos: Análisis y diseño", Segunda Edición.
- Stallman Richard M(2004). ; "Software libre para una sociedad libre", Primera Edición.

- Valenzuela Ruz,Víctor(2003)."Análisis y Diseño de Algortimos", Primera Edisión. Copiapó,Chile.
- Hernandez, M. y Manzanares, J.(2002)."Protección Jurídica del Software". Universidad de Málaga.

Referencias Electrónicas

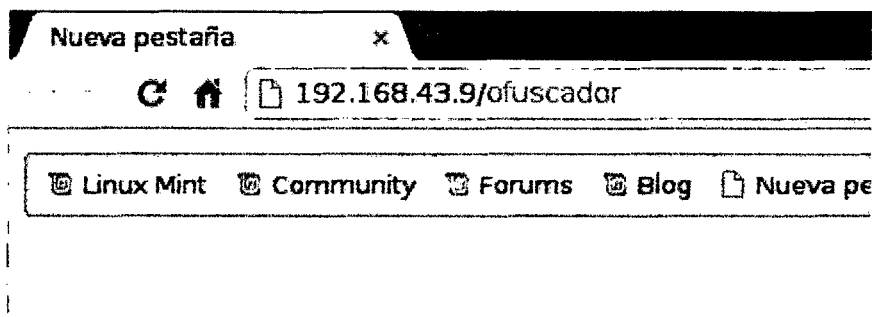
- Ministerio de Relaciones Exteriores del Perú, Propiedad Intelectual, Marcas Comerciales y Patentes, disponible en http://www.rree.gob.pe/portal/economia2.nsf/0/083f01d77ce65361052569bc005b8e3f?OpenDocument, consultado el 07 de agosto 2013.
- Organización mundial de la protección intelectual, disponible en http://www.wipo.int/about-wipo/es/what_is_wipo.html, consultado el 07 de agosto 2013.
- Thomson Ivan, "Satisfacción del Cliente", disponible en http://www.promonegocios.net/mercadotecnia/satisfaccion-cliente.htm, consultado el 07 de agosto 2013.

ANEXOS

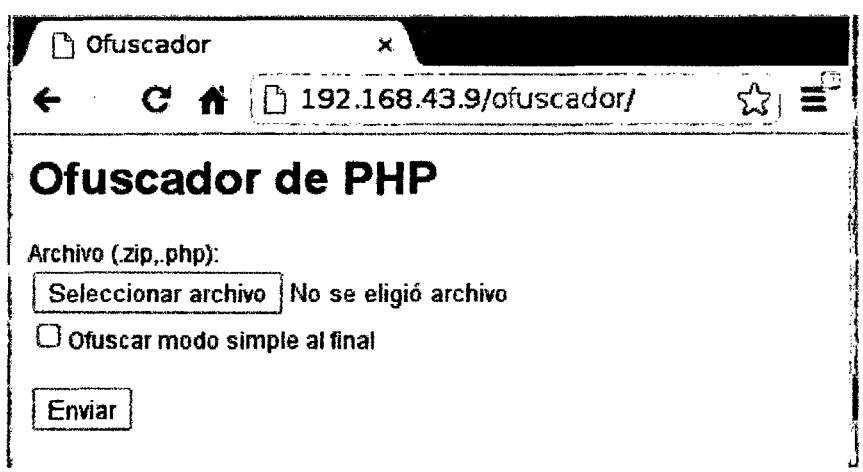
ANEXO A

MANUAL DE USUARIO

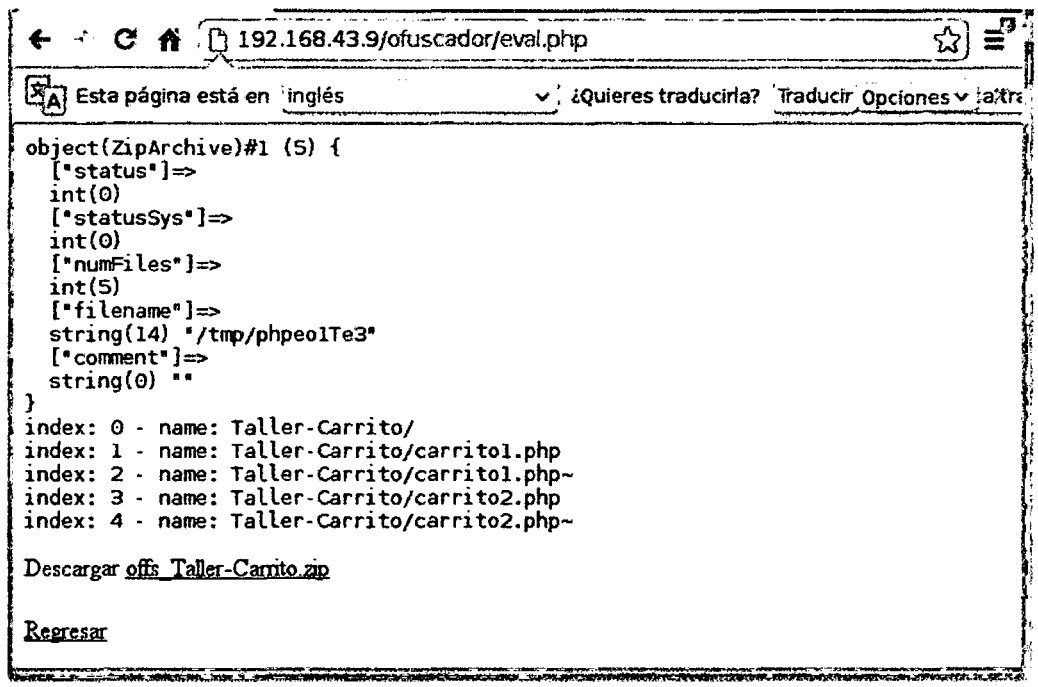
- **Primero :** Una vez implantado el algoritmo en el servidor con la tecnología LAMP, se realiza el acceso de la siguiente manera.



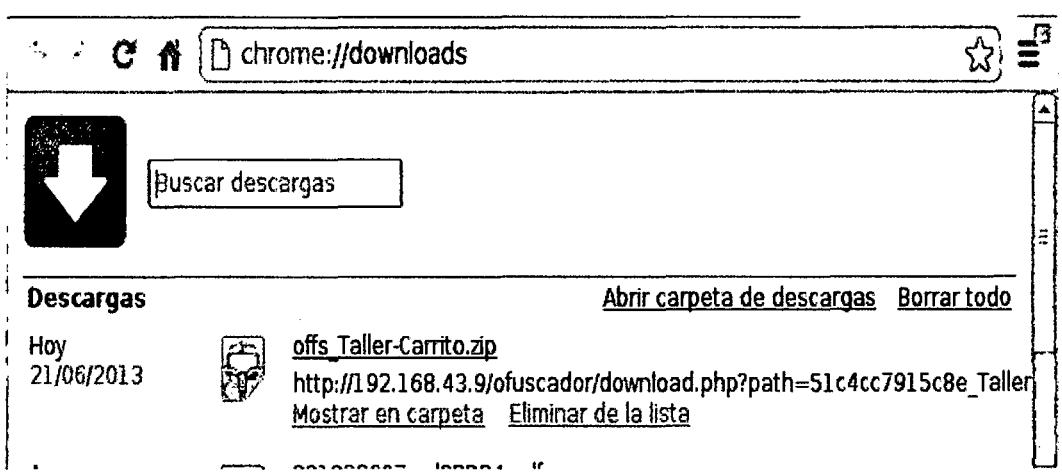
- **segundo**: La ejecución del algoritmo de ofuscación muestra el formulario de entrada de datos, en la cual permite la selección del archivo .php a ofuscar o un software empaquetado en .zip.



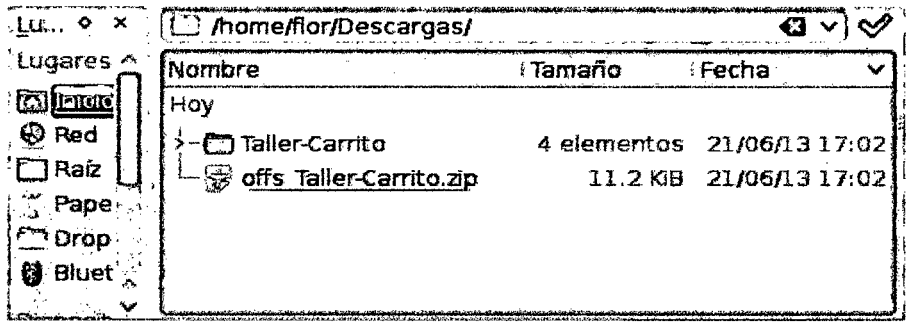
- **Tercero:** Seleccionamos el archivo .php a ofuscar o un software empaquetado en .zip, se hace click en enviar teniendo como los archivos .php ofuscados.



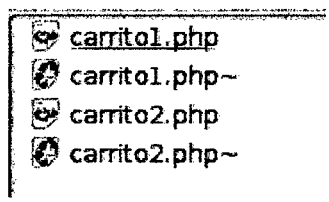
- **Cuarto**: Descargamos el archivo ofuscado.



- **Quinto:** Descomprimimos el archivo ofuscado, y se observa que el archivo se descomprime en su nombre original como se muestra en la siguiente figura.



- **Sexto:** Abriendo la carpeta del software ofuscado se observa que los archivos no han cambiado su nombre y que el contenido ha sido ofuscado.




```
$a521625ac9aa97 =
"Qlu4KoY4JfJFAo3Zc0jDttQboowupOJLUXkkOACgY836/k/30bkn/vVFoQXHTTJFVZLgZBJpoC4EkOxj2JoCgc2FVhpDsHiQcys7Lb
bkabqJBkWstZ/GyoCBHsgKCnMXFC6F7xEhUlrW29VizAWRrql/+ibbHt6XmGwskP+qeltd3VnWXgWkXDy5g3cbnEd5WfztxHJfKUKR7.
Jr3Ss58lrv+w5sWrINnd8fmJfu4+w8rbDSXoWl3relmOXVVJZnE/cDK2lGzbU9azrnx2TivIL/rH6LQS+bNc3IKjrf9bW5l7l7Y9u+Hf
vPVdA55cPyObquRE480sdJZCllozsJ17P7IchsZfsUrbrVcHwp5P2oPz8HgOy2s30H88/TVp9swbjL++V4eL8PjfCx905hMdquY/tWz
7+rU7Lr7vm335kDM7zsQameXPwfeW/eDxGYun2bpf008j8tXeM8reex3PB+3s4WV3o8ru055P5/Wz2lmltsvovu5+vMze7haXrcNAqs
h4lO+dC6D19RLIrX8nbrxG6aePeatVdjOa5aIzE9Z7e3wzJrvdZNslnZtl+O5Qwqk2z4CTu3+/E6lX4zGr9Dua+2relwYGwOs2XWGbz
HlWlf3DagRzv9k+zNezsD05t8R4e2l/
nE63ahEXN3vfqMrHbL4jVXaoT7LuWPt8ss9sDlNnb8+agXaybklj6+Slt5X3Vl8YyWXePwhPPePiI6pWG3l5de7XXVA8JdXK+nGzPTv
okOazVlt5nm490Zx0HbzavD5+0cSsoIfLOwvllWYuWW+WU2+JydTnLZf3bP7iFelaLrEJSf3btrq2H4/uTX16PhteXrZJbdnl2wfg23
vXaZt0jyP3ufTD2671rowz8WtYWAbBcxo7j7G5bEXOo7lzHsNoaC2bZfFZRR0vO03ifX/feQ5ldX98x6+NfIzLwafraudQVorWsbXZJ
Mj/IYTgf7zPS4N0T+lXwKz+Yh6kza7vwy37n0dVYxK53b+Da5P9t+9rL1L+tL693aW8dnsIvC/nUSC+vxdCfXYafb7sbD7VPcnuOet/
VWjnDHD6HDBx+IQC+06kahiE6jsM7qPqMwCRMozWG2409aIZWiT8gDwW/2XPIkFgUoUdfqtxC9wthQaE41JDcok8xgRwbqfAORd8aSdl
$c521625ac9ad5f =
```

- **Séptimo**: Ejecutamos el software ofuscado obteniendo la misma funcionalidad del software.

ANEXO B

TEST DEL ALGORITMO DE OFUSCAMIENTO

Archivo: **sistema.php //antes de ofuscar**

```
<?php
	session_start();
	$CARRITO=$_SESSION['SESIONCARRITO'];
		if (isset($_POST['idprod']))
		{
			$id=$_POST['idprod'];
			$producto=$_POST['nombreprod'];
			$costo=$_POST['costoprod'];
			$cantidad=$_POST['cantidadprod'];
			$encontrado=0;
				if (!isset($CARRITO)){
					$CARRITO[$id][1]=$id;
					$CARRITO[$id][2]=$producto;
					$CARRITO[$id][3]=$costo;
					$CARRITO[$id][4]=$cantidad;
					$CARRITO[$id][5]=$costo*$cantidad;
					}
				else
					{foreach($CARRITO as $k => $v)
						{ if ($id==$k)
							{
						$CARRITO[$id][4]+=$cantidad;
							$CARRITO[$id]
[5]+=$CARRITO[$id][3]*$cantidad;
							$encontrado=1;

							}
						}
					if ($encontrado==0) {
						$CARRITO[$id][1]=$id;
						$CARRITO[$id]
[2]=$producto;
						$CARRITO[$id][3]=$costo;
						$CARRITO[$id]
[4]+=$cantidad;
						$CARRITO[$id]
[5]+=$costo*$cantidad;
						}
					}
		}
		$_SESSION['SESIONCARRITO']=$CARRITO;
		?>
			<table width="60%" cellspacing="0" cellpadding="2"
border="1">
```

```
                <tbody>
                        <tr>
                        <form  action="<? $PHP_SELF ?>" method="post">
                                <td valign="top">
                                <table cellspacing="2" cellpadding="2" border="0">
                                <tbody>
                                                <tr>
                                <td align="center">
                                 </td>
                                <td valign="top">
                                <br>
Id :<input type="text"  name="idprod"><br>
        Nombre: <input type="text"  name="nombreprod"><br>
        Cantidad: <input type="text"  name="cantidadprod" size="5px"><br>
        Costo: <input type="text"  name="costoprod"><br><span class="">
        <button type="submit" id="tdb4" class="ui-button ui-widget ui-state-default
ui-corner-all ui-button-text-icon-primary ui-priority-secondary" role="button"
aria-disabled="false">
<span class="ui-button-icon-primary ui-icon ui-icon-refresh"></span>
<span class="ui-button-text">Actualizar</span>
        </button>
        </span>
        </td>
        </tr>
        </tbody>
        </table>
        </td>
<td valign="top" align="right"><strong>hhhh</strong></td>  </form></tr>
                                </tbody>
                        </table>
<?php
if (isset($CARRITO))
{       foreach($CARRITO as $k => $v)
         {
                echo "id : ".$v[1]."<br>";
                echo "Producto : ".$v[2]."<br>";
                echo "Costo : ".$v[3]."<br>";
                echo "Cantidad : ".$v[4]."<br>";
                echo "Costo total : ".$v[5]."<br>";
         }
}
?>
```

Archivo: **sistema.php // ofuscado en modo optimizado**

```
<?php
$a521799c77f759 = "/DKe6YnBOGAbw8Cvvy2pwR7dBxdgoW61dHR0uj//+8/n";
$a521799c780f8f =
"\102\x49\165\x4a\151\67\157\x6b\101\147\61\x39\x6e\x39\x5a\112\x61\x33\160\x
55\162\53\126\x55\x68\66\x7a\x2f\x2b\x46\x56\146\x64\145\62\x31\141\131\150\x
50\124\x38\71\x65";
$a521799c7814fc = "34GkVpA3Y5B3G5AgASkiQGpYkDdekIIIJFuA1LkgL0og";
$a521799c7816e6 =
"\105\163\x6a\x52\x41\x46\114\153\x67\162\x79\x6c\x67\x4c\167\x72\x67\x6c\x51
\x45\x49\106\x39\x63\x51\142\x35\153\147\156\172\132\x41\x4d\x6e\172\111\111
\x6b\x58\65\x50\60\x58";
$b521799c78011e =
"\x32\x73\x32\x36\157\x72\62\151\x65\117\143\x65\x34\150\x64\116\104\105\x59\
x68\170\105\157\150\151\103\60\x43\x41\122\x45\160\x58\x45\117\144\x75\103\x
58\127\x68\x54\163\x75";
$b521799c780d93 =
"\165\x45\110\156\x4c\152\x57\101\x7a\x31\x6b\x6d\105\57\x39\x6d\x50\106\x71\
x77\150\x61\170\x50\152\x59\110\70\x57\x4b\120\63\x61\153\117\x74\x70\x55\10
6\105\x4d\x45\120\53";
$b521799c78157d =
"CQ6kQgT5wgCyOUGKK5CDCuRFBKTEBfnAA1JrglQFIGU0";
$c521799c77fc72 = "FRDPUsfPOH56WNyud61yzXubiiFhNXlQ4XQ8uWni3yke";
$c521799c7804e2 = "VoXlednuC3vg+dy/avsng+nd9nnXXlc+4cmMETAMHjwt";
$c521799c7813fc = "HG/y3tsdyPPdGZsSkULjrWJNhO1OCQ5P5d0H4f5BBPZB";
$d521799c78000e = "FmdX16o3LsuuPeiK7LZ49oPh83B1J+xvzIQIUMhKRCUF";
$e521799c7810c8 =
"\161\154\145\x2f\x2f\65\160\154\x75\61\x62\x59\156\x55\154\x70\x70\x35\x35\x5
3\122\104\171\170\x66\x6c\152\123\x79\102\x4d\64\x59\61\x67\70\131\x31\154\1
14\x34\144\x75\125";
$e521799c7812a9 =
"\61\x2b\x48\x70\124\164\143\x65\x4b\x6a\67\x61\x33\x6f\71\142\164\66\x65\x79\
155\71\163\x39\164\114\x2b\x75\x61\x31\132\x59\x38\166\x66\x74\143\x71\144\7
1\156\x6b\x30\146";
$f521799c781311 =
"\x64\67\x72\62\153\116\145\125\71\160\x76\71\x31\x2b\116\x4a\x31\150\57\x4f\x
62\64\123\x39\x71\117\172\66\x68\x6f\70\x30\x70\x2f\x6b\x30\x70\115\131\172\1
66\x68\166\105";
$f521799c7815fe =
"\123\x44\x49\107\123\x59\101\x67\x48\70\71\101\125\151\104\x49\171\x79\x44\1
11\x79\x7a\x78\x49\x65\x67\101\x35\107\60\x47\x2b\110\x6f\112\70\156\121\114\
65\x34\147\x54\171";
$g521799c780097 =
"\x55\x5a\x4a\x36\x79\x78\x74\x32\x4c\x44\x61\x61\x45\143\x35\x6c\x55\123\x59\
167\170\156\x39\x57\152\150\131\x67\x4d\105\x5a\x6f\113\x2b\x57\65\110\125\x
32\x50\x66\172\157\x36";
$h521799c78019a =
"\x31\123\131\165\105\x51\x75\x4d\x41\113\115\x50\x67\163\x73\113\x35\x39\x7
0\126\61\x49\x44\67\x4a\170\146\x32\x61\x6a\101\67\103\x4f\x4f\x75\x50\121\x5
4\124\x53\60\61\151";
```

```
$h521799c780ffd =
"\x66\57\131\x75\65\x37\x76\x50\132\61\61\x37\x4b\106\147\x4b\x4c\x6d\x63\x48\
167\x2b\156\x33\x67\70\145\x37\x6c\142\104\x58\x44\106\131\x31\171\x57\x36\5
3\130\146\x6a\x36";
$i521799c77f6e9 =
"ZVhdUxvJDn3fX7G1dR+SSpWY7ukPT+3eTQHhM5AQbCC4";
$i521799c77fa50 =
"\67\x6e\x6f\132\162\144\67\x36\x75\x49\114\x4c\x4f\x55\101\x54\120\172\141\15
2\130\126\122\x6d\x4d\123\142\x36\127\x78\x6b\156\x76\x4a\x75\x43\x6a\x2b\x5
8\x69\61\x6b\x61\150";
$i521799c77fe83 =
"\x45\x56\116\x78\111\x52\x64\172\64\x62\101\71\x34\x6d\x6e\x79\152\x50\120\x
6a\143\x58\x30\70\117\127\106\61\x73\x6f\106\x56\110\x54\x67\x76\x56\105\61\x
4c\61\127\166\x7a";
$i521799c780439 =
"1UPvu3VFc1WkCWmYxzAETJ95S3hBYRFD5Vp4l4E68GZ8";
$i521799c780973 = "ljRCMuZD5buuIcsSzIk7mBGIGUgU9hjupiYKZsLccLYl";
$i521799c7817f2 =
"\102\x73\147\63\141\112\x43\166\x4d\171\x42\x31\117\153\151\x6c\x42\120\112\
103\x42\x33\114\x32\x67\x64\x54\x79\111\x42\x55\165\x53\111\153\x44\70\152\x
30\106\65\x44\x73\x68";
$j521799c780eb1 = "mRfeZcECemf+dHDrjmfTrj3EVdaiw/Bk40O6JD4Qeoqn";
$k521799c7802a2 =
"\121\147\172\x50\x6c\x6c\110\104\103\x75\x2f\147\x77\x33\165\104\161\x69\x7a
\x74\150\167\146\114\143\x69\131\114\114\x45\166\155\117\x42\x37\x72\165\16
4\x36\x31\65\160\117\x64";
$k521799c7806b5 =
"\66\170\x6f\171\x52\61\x73\x66\x68\x4c\x4c\60\x30\x2b\71\x33\160\x78\x4f\x30\x
33\66\53\x58\172\153\x6f\53\x57\105\x77\132\x4b\121\60\141\x6a\x48\x51\64\x52
\65\126\x74";
$k521799c7809f4 =
"\x37\144\x76\146\63\x65\x2b\x66\x50\x6c\x78\115\144\147\x6a\x46\x69\x77\60\1
70\x31\x31\x43\x35\155\104\111\x64\x7a\x52\141\143\66\x66\x47\157\107\x4b\x5
7\171\164\101\166\x43";
$l521799c78031e = "H8+fzfIk+PK7aw8mp7Sp5YUq756/nwS0TzeA2nDsBHWM";
$l521799c781661 =
"QRLkyAQpQkG+loC8rYG8xYCURiC1EMjnBZDvCSCpF+Qb";
$m521799c77f851 = "l1qjfWd++piZ2a/tCu3jtb03dmj/cpQdNV7vHvW79qBC";
$m521799c781220 =
"\x44\102\117\106\x59\164\160\x63\117\64\146\143\125\x73\x31\155\106\60\57\x
44\x32\145\x54\70\x59\130\x69\116\x39\x70\x63\x4e\x2b\x63\65\64\115\157\x50\x
62\63\141\x50\112";
$n521799c77f9dc = "W1vGjxopr6JHEZfV0Wbid6brHXQe6GDjw1+Lm5+7X+bC";
$n521799c781483 =
"\x36\155\x4b\121\164\x32\171\121\x31\62\161\x51\131\150\152\x6b\x4e\x51\x33\
x6b\107\x77\x4e\111\170\121\110\171\x5a\121\x69\153\x30\101\122\x35\x48\x77\
x66\65\x57\101\116\x53";
$o521799c77fd75 =
"\x31\x65\x46\x58\70\x32\x6b\65\x36\x4e\x70\x44\161\x4b\151\x4c\57\x50\63\151\
57\155\112\x57\126\x69\163\162\x37\111\x32\170\x37\60\126\162\104\x5a\66\x2b
```

```
\x6f\166\x4b\x33";
$o521799c781132 = "RFzmdqe/tXO53N/q2oP3b/Jb0a2CJj1dU10CUMVDCInX";
$p521799c77fb50 =
"\161\124\x72\x32\x62\x6a\157\x50\x43\x32\105\166\144\115\65\x5a\x4d\127\162\
x6b\x59\157\x67\165\x4a\151\106\x54\x47\105\x58\143\x4b\64\x62\165\131\65\11
2\x38\x79\x61\x45\x72";
$p521799c7808e0 =
"\57\157\x30\x34\106\x63\x35\x6c\121\x2f\x56\161\164\162\x39\144\x75\x4c\x33\1
51\x64\156\66\x41\x39\166\x6b\107\62\x6c\123\x61\x2f\63\x46\150\x78\53\157\x4
e\147\x2f\x4c\115";
$q521799c77f8d4 =
"\x6a\x76\x62\x36\x62\114\142\71\x64\126\x6b\x4f\102\x76\166\x43\130\x71\x6f\1
13\67\x64\156\x73\125\x2f\53\x36\x48\132\x51\166\107\x64\161\162\x64\x58\170\
x57\x6b\x33\62\153";
$q521799c78021e = "CQQEBI2FMgQOYI9sPwIKgeF2gzy0fk8othEI0Tg6GXE1";
$q521799c78185e =
"\x79\x49\144\102\105\x41\57\x4c\x48\x7a\71\x2b\57\120\165\120\x7a\57\57\x2b\
70\57\154\57";
$r521799c77f944 = "lB0p50atzUfK5CP8NXf0lfOj+GWb265rbUPJrpkftZ5d";
$r521799c77face =
"\163\130\x34\x33\152\160\x75\x51\x6c\x72\131\106\146\64\x7a\x6d\x70\130\125\
x38\x68\x47\65\x47\x75\x46\166\143\126\x56\153\124\x66\62\132\144\65\x7a\162
\63\x35\120\x7a\x6a";
$r521799c780621 = "Nbs+e916eUL7741hp9KIV8QutHNLzNAbRYImreBHREW6";
$r521799c7807c4 = "xDysE6NcgkQcZ8plVFwaWEjKNOYq2p2isj3hXVn7zi6o";
$s521799c780c72 = "ae8fDivbtQeXve1ueffMca8rTbSLrYGqBKFicuHcM1SS";
$s521799c780e30 =
"\x38\x43\171\x75\x73\102\164\167\105\121\123\x43\64\x59\x54\57\x6e\63\120\14
4\x30\60\x30\151\x38\64\170\106\125\x43\x52\164\x51\x72\x56\114\70\x67\x43\1
04\172\x33\121\x71";
$s521799c781068 = "6gbtz+v94wYJjIpnPGoQU3I/+3TpQ5UfcKZ2nWvrSOs9";
$u521799c77fbd9 =
"\x4b\x35\61\x4c\x63\x6d\66\x33\130\x71\145\110\145\64\166\x32\60\x77\53\x30\
150\x33\x58\x6b\126\x55\x4e\157\x73\152\57\120\144\150\x38\x48\162\150\x78\1
14\x65\x31\62\116";
$u521799c780d1e = "+f6l31uZs2Ej7LXPE1QU9TGeCrVVngASBwG1YOQdxe33";
$u521799c78119b =
"\x6b\x4f\x52\161\x4e\x2b\151\141\x68\145\124\106\131\x2f\x39\x31\x59\x50\x4b\
x36\x58\x41\x6c\67\160\x4a\x67\105\x47\x4d\x57\x39\147\x32\106\156\x66\105\6
3\103\126\107\x42\x71";
$v521799c77fc00 = "08mwmvEY1vPJsBwmnjTvxV3T99p2vSvzFro2HLeKhbQu";
$v521799c77fe0b =
"\163\x52\117\125\71\x68\123\57\x63\160\x5a\x67\x77\x61\110\x2f\x57\x6d\70\x2
b\114\x75\156\147\165\x2f\x72\x79\x35\x69\104\x37\146\146\126\70\x30\162\x55\
x48\x33\66\x54\x46";
$v521799c77ff84 = "GRuaoR9DJOwGXBOjesPqhm/C2rbpF+XR88vyXtiDoYpe";
$w521799c780f28 =
"\162\x46\142\103\62\65\124\x55\62\x31\165\x7a\x36\x65\172\x54\x63\x62\x34\x7
a\120\x52\130\x32\143\143\x46\160\x77\x33\167\152\123\141\x41\x34\60\x53\167
\x6b\x38\101\x71\103";
```

```
$w521799c781767 =
"\x70\x4a\x51\101\x2b\x52\x77\103\x55\156\x4f\104\x6c\x49\x73\x67\156\x31\102\
102\x79\x6e\x43\121\x73\x78\110\153\127\170\x56\111\107\121\x6c\x53\x6b\111\
x4d\x55\106\151\x43\156";
$x521799c78085d =
"\x64\x6a\101\x6c\x6a\x75\164\x42\120\x31\x51\141\x2b\x7a\147\x34\70\161\65\x
76\x72\x5a\x71\x51\x6b\x47\x78\111\145\x4e\101\x61\x57\x45\x54\x73\65\x39\15
1\x37\166\x4c\x4e\102";
$x521799c780ac7 = "iOXWhmevSkTuCsY/UjJSEP2OsynjY6IAdUp4jxUR5urL";
$y521799c77ff14 = "nKrLNYmtEDONFVYq63rX3lBeto+2FlVm7heVsl+9TXjJ";
$y521799c7803a4 =
"\107\126\120\x72\x4c\x64\x55\x57\104\65\x39\167\123\x2f\x32\153\146\144\143\
x35\x5a\x4d\107\163\125\x59\x36\131\61\x6a\161\170\x59\x45\67\x63\124\x4c\x7
9\x43\x76\105\115\115";
$y521799c780583 =
"\x78\153\x73\70\152\x43\x78\154\x4f\x51\111\145\x73\x72\120\x31\x36\x73\x34\x
53\x37\x36\x30\117\126\x37\x30\x64\x2b\x2f\x4f\x77\142\142\162\x32\x32\x6e\x6
9\113\x37\166\x68\x77";
$y521799c78137b =
"bueWM92XjUY3acBjW9PdJmmHPA08i5U1PFiwDq5hlWuE";
$z521799c77f7d4 =
"\150\x38\x6e\104\110\57\x38\160\x72\126\141\53\x4b\103\162\166\x4b\66\x32\1
41\120\x2f\x2f\x37\65\x31\71\x35\125\x2b\x64\x31\x38\x35\x6f\63\x33\57\57\x36\1
45\71\x4e\145";
$z521799c77fd04 =
"\66\64\126\x33\157\62\x76\x32\172\x6d\x4f\x6b\x50\156\x6f\x37\162\147\156\x74
\110\127\x76\123\x2b\x6c\x35\x61\154\x57\161\105\63\x73\x30\x36\143\x6c\71\x3
7\67\161\112\x66";
$z521799c780734 =
"\71\132\x68\x52\x61\x58\x42\x31\x4a\142\170\172\112\166\x52\106\x56\162\71\x
73\61\x39\x65\x44\160\53\x4f\165\x50\x56\x52\143\116\x4c\x2b\x71\102\x37\x6c\
53\117\116\x6d\x6a";
$z521799c780b77 = "l6vZwf21OxX2imnF1cN8Ug8W+w9oX270h1Zpd8NMjuF6";
$z521799c780bee =
"\x37\156\126\153\105\x6b\167\x4d\141\167\130\67\x74\166\x75\107\144\70\127\
170\x66\x79\155\117\x39\60\x34\130\x4c\170\x50\x33\x54\144\x68\x72\x51\x31\x
6b\x35\x2b\x58\61\x31";
$e521799c7822f6 = "\142\x61\163\145\66\64\x5f\x64\145\143\157\x64\x65";
$h521799c782375 = "\x67\x7a\151\156\146\154\x61\164\x65";
eval($h521799c782375($e521799c7822f6($i521799c77f6e9.$a521799c77f759.$z
521799c77f7d4.$m521799c77f851.$q521799c77f8d4.$r521799c77f944.$n521799
c77f9dc.$i521799c77fa50.$r521799c77face.
$p521799c77fb50.$u521799c77fbd9.$v521799c77fc00.$c521799c77fc72.$z5217
99c77fd04.$o521799c77fd75.$v521799c77fe0b.
$i521799c77fe83.$y521799c77ff14.$v521799c77ff84.$d521799c78000e.
$g521799c780097.$b521799c78011e.$h521799c78019a.$q521799c78021e.
$k521799c7802a2.$l521799c78031e.
$y521799c7803a4.$i521799c780439.$c521799c7804e2.$y521799c780583.$r521
799c780621.$k521799c7806b5.$z521799c780734.$r521799c7807c4.$x521799c7
8085d.
```

```
$p521799c7808e0.$i521799c780973.$k521799c7809f4.$x521799c780ac7.$z521
799c780b77.$z521799c780bee.$s521799c780c72.$u521799c780d1e.
$b521799c780d93.$s521799c780e30.$j521799c780eb1.$w521799c780f28.$a521
799c780f8f.$h521799c780ffd.
$s521799c781068.$e521799c7810c8.$o521799c781132.$u521799c78119b.
$m521799c781220.$e521799c7812a9.$f521799c781311.$y521799c78137b.
$c521799c7813fc.$n521799c781483.$a521799c7814fc.$b521799c78157d.
$f521799c7815fe.
$l521799c781661.$a521799c7816e6.$w521799c781767.$i521799c7817f2.$q521
799c78185e)));
?>
```

ANEXO C

CÓDIGO FUENTE DEL ALGORITMO DE OFUSCACIÓN

index.php

```
<html>
<head>
<title>Ofuscador</title>
<style>
	body{ font: 12px Arial; }
</style>
</head>
<body>
	<h1>Ofuscador de PHP</h1>
	<form action="eval.php" method="post"
enctype="multipart/form-data">
		<label for="file">Archivo (.zip,.php):</label><br/>
		<input type="file" name="file" id="file" /><br/>
		<input type="checkbox" name="check" id="check" /><label
	for="check">Ofuscar modo simple al final</label><br/><br/>
		<input type="submit" value="Enviar" />
	</form>
</body>
</html>
```

eval.php

```
<?php
	include("ofuscador.php");

	$retorna =  '<br><br><a href="index.php">Regresar</a>';
```

```
	$noinc = array("config.php","tcpdf/","excel/");

	echo "<pre>";
	if ($_FILES["file"]["error"] > 0) die("Error: " . $_FILES["file"]["error"] .
"<br>".	$retorna);

	/* $_FILES["file"][___] "name" "type" "size" "tmp_name" */

	$ext = GetEXT($_FILES["file"]["name"]);
	//print_r($_FILES);
	//print_r($_POST);
	//echo $_FILES["file"]["name"];
	//echo "#".$ext."#";
	$modosimple = isset($_POST["check"])?true:false;

	if($ext=='php'){
		$ftmp = uniqid()."_".$_FILES["file"]["name"];
		$fh = fopen("./tmp/".$ftmp, 'w') or die("no se puede crear
archivo");
		$fcont = file_get_contents($_FILES["file"]["tmp_name"]);
		$fcont = Ofuscar($fcont,$modosimple);
		fwrite($fh, $fcont);
		fclose($fh);
		echo 'Descargar <a href="download.php?path='.
			$ftmp.'&name=offs_'.$_FILES["file"]["name"].'">offs_'.
$_FILES["file"]		["name"].'</a>';
		echo $retorna;
		exit(0);
	}

	if($ext!='zip')die("Archivo no es zip ni php <br>".$retorna);

	$za  = new ZipArchive();
```

```
$zip = new ZipArchive();

$fzip = uniqid()."_".$_FILES["file"]["name"];
$filename = "./tmp/".$fzip;

if($zip->open($filename, ZIPARCHIVE::CREATE)!==TRUE) die("no se
puede crear archivo zip resultado".$retorna);

if($za->open($_FILES["file"]["tmp_name"])!==TRUE) die("no se puede
abrir archivo zip subido".$retorna);

var_dump($za); // var_dump
echo "numFicheros: " . $za->numFiles . "\n";
echo "estado: " . $za->status  . "\n";
echo "estadosSis: " . $za->statusSys . "\n";
echo "nombreFichero: " . $za->filename . "\n";
echo "comentario: " . $za->comment . "\n";


for ($i=0; $i<$za->numFiles;$i++) {
        $if = $za->statIndex($i);
        $fpath = $if['name'];

        echo "index: $i - name: ".$fpath."\n";

        if(!preg_match("/\/$/",$fpath)){
                $fcont = $za->getFromName($fpath);
                $ext = GetEXT($fpath);

                if($ext == 'php'&&NoArray($noinc,$fpath) ){ /
                        $fcont = Ofuscar($fcont,$modosimple);
                }else if($ext == 'js'){
                        $fcont = MinJS($fcont); //Ofuscar JS
                }else if($ext == 'css'){
```

```
                        $fcont = MinCSS($fcont); //Ofuscar CSs
                    }

                    $zip->addFromString($fpath, $fcont); /
                }
            }

            $za->close();
            $zip->close();
            echo "</pre>";

            echo 'Descargar <a href="download.php?path='.$fzip.'&name=offs_'.
            $_FILES["file"]["name"].'">offs_'.$_FILES["file"]["name"].'</a>';

            echo $retorna;

?>
```

ofuscador.php

```
<?php

function NoArray($arr,$fpath){
        foreach($arr as $cod=>$val){
                if(preg_match("/".str_replace('/','\/',$val)."/i",$fpath))return false;
        }
        return true;
}
function MinCSS($str){ //Minimizar CSS
        return $str;
}
function MinJS($str){ // Minimizar JS
        return $str;
}
```

```
function GetEXT($fpath){
	$ext = explode('.',$fpath);
	$ext = strtolower($ext[count($ext)-1]);
	return $ext;
}
function MinHTML($str){
	$search = array(
		'/\>[^\S ]+/s',
		'/[^\S ]+\</s',
		'/(\s)+/s',
	);
	$replace = array(
		 '>',
		'<',
		'\\1',
	);
	$content = preg_replace($search, $replace, $str);
	return $content;
}

function foct($char){ //convertir caracter a OCT
	return decoct(ord($char));
}

function fhex($char){ //convertir caracter a HEX
	return dechex(ord($char));
}

function fstr($str){ //covertir cadena a HEX y OCT
	$tstr = "";
	for($i=0;$i<strlen($str);$i++) $tstr .=
(rand(0,1)?'\\'.foct($str[$i]):'\\x'.fhex($str[$i]));
	return $tstr;
}
```

```
function OfPHP($content){
 "eval(gzinflate(base64_decode("
        $minstr = $content;
        $minstr = preg_replace("/'.*?'/","",$minstr);
        $minstr = preg_replace('/".*?"/',"",$minstr);
        $n1 = substr_count($minstr,'<?');
        $n2 = substr_count($minstr,'?>');
        if($n1>$n2){$content .= "?>";}
        $content = preg_replace("/<<<S(.*?)S;/","<<<EOT\n$1\nEOT;\n",
$content);
        $content = preg_replace("/<<<EOT(.*?)EOT;/","<<<EOT\n$1\nEOT;\n",
        $content);
        $content = str_replace('.$',' .$',$content);
        $content = base64_encode(gzdeflate('?>'.$content.'<?'));
        $tam = strlen($content);


        $npartes = $tam<100?rand(2,10):rand(50,100);


        $partes = array();

        $ptam = round($tam/$npartes);

        $vars = array();
        for($i=0;$i*$ptam<=$tam;$i++){
                $uid = chr(rand(ord('a'),ord('z'))).uniqid();
                $partes[$uid] = substr($content,$i*$ptam,$ptam);
                $vars[] = $uid;
        }

        $final = '$'.implode('.$',$vars);
```

```
        ksort($partes);
        $stringData = "";
        foreach($partes as $cod=>$val){
                $stringData .= '$'.$cod.' = "'.(rand(0,1)?fstr($val):$val).'";'."\n";
        }

        $ideco = '$'.chr(rand(ord('a'),ord('z'))).uniqid();
        $stringData .= $ideco.' = "'.fstr('base64_decode').'";'."\n";

        $igzin = '$'.chr(rand(ord('a'),ord('z'))).uniqid();
        $stringData .= $igzin.' = "'.fstr('gzinflate').'";'."\n";

        $stringData .= 'eval('.$igzin.'('.$ideco.'('.$final.')));';l
        return "<?php\n".$stringData."\n?>";
}


function OfPHPs($str){
        $ideco = '$'.chr(rand(ord('a'),ord('z'))).uniqid(); //variable con hex y oct
para la funcion base64_decode
        $strd .= $ideco.' = "'.fstr('base64_decode').'";'."\n";

        $igzin = '$'.chr(rand(ord('a'),ord('z'))).uniqid();
        $strd .= $igzin.' = "'.fstr('gzinflate').'";'."\n";
        $minstr = $str;
        $minstr = preg_replace("/".*?"/","",$minstr);
        $minstr = preg_replace('/".*?"/',"",$minstr);
        $n1 = substr_count($minstr,'<?');
        $n2 = substr_count($minstr,'?>');
        if($n1>$n2){$str .= "?>";}
        $str = preg_replace("/<<<S(.*?)S;/","<<<EOT\n$1\nEOT;\n",$str);
        $str = preg_replace("/<<<EOT(.*?)EOT;/","<<<EOT\n$1\nEOT;\n",
$str);
        $content = str_replace('.$',' .$',$content);
```

```
        $strd = "<? eval($igzin($ideco('".base64_encode(gzdeflate('?>'.$str.'<?
php')).'')))); ?>";
        return $strd;
}

function Ofuscar($content,$mds){
        $content = compress_php_src($content);
        $content = MinHTML($content);

        $r = rand(1,3);

        for($i=0;$i<$r;$i++) $content = OfPHP($content); /
        if($mds==true)$content = OfPHPs($content);
        return $content;
}



function compress_php_src($src) {

    static $IW = array(
        T_CONCAT_EQUAL,            // .=
        T_DOUBLE_ARROW,            // =>
        T_BOOLEAN_AND,             // &&
        T_BOOLEAN_OR,              // ||
        T_IS_EQUAL,                // ==
        T_IS_NOT_EQUAL,            // != or <>
        T_IS_SMALLER_OR_EQUAL,     // <=
        T_IS_GREATER_OR_EQUAL,     // >=
        T_INC,                     // ++
        T_DEC,                     // --
        T_PLUS_EQUAL,              // +=
        T_MINUS_EQUAL,             // -=
        T_MUL_EQUAL,               // *=
```

```
    T_DIV_EQUAL,            // /=
    T_IS_IDENTICAL,         // ===
    T_IS_NOT_IDENTICAL,     // !==
    T_DOUBLE_COLON,         // ::
    T_PAAMAYIM_NEKUDOTAYIM,  // ::
    T_OBJECT_OPERATOR,      // ->
    T_DOLLAR_OPEN_CURLY_BRACES, // ${
    T_AND_EQUAL,            // &=
    T_MOD_EQUAL,            // %=
    T_XOR_EQUAL,            // ^=
    T_OR_EQUAL,             // |=
    T_SL,                   // <<
    T_SR,                   // >>
    T_SL_EQUAL,             // <<=
    T_SR_EQUAL,             // >>=
);
if(is_file($src)) {
    if(!$src = file_get_contents($src)) {
        return false;
    }
}
$tokens = token_get_all($src);

$new = "";
$c = sizeof($tokens);
$iw = false; // ignore whitespace
$ih = false; // in HEREDOC
$ls = "";    // last sign
$ot = null;  // open tag
for($i = 0; $i < $c; $i++) {
    $token = $tokens[$i];
    if(is_array($token)) {
        list($tn, $ts) = $token; // tokens: number, string, line
        $tname = token_name($tn);
```

```
            if($tn == T_INLINE_HTML) {
                $new .= $ts;
                $iw = false;
            } else {
                if($tn == T_OPEN_TAG) {
                    if(strpos($ts, " ") || strpos($ts, "\n") || strpos($ts, "\t") || strpos($ts,
"\r")) {
                        $ts = rtrim($ts);
                    }
                    $ts .= " ";
                    $new .= $ts;
                    $ot = T_OPEN_TAG;
                    $iw = true;
                } elseif($tn == T_OPEN_TAG_WITH_ECHO) {
                    $new .= $ts;
                    $ot = T_OPEN_TAG_WITH_ECHO;
                    $iw = true;
                } elseif($tn == T_CLOSE_TAG) {
                    if($ot == T_OPEN_TAG_WITH_ECHO) {
                        $new = rtrim($new, "; ");
                    } else {
                        $ts = " ".$ts;
                    }
                    $new .= $ts;
                    $ot = null;
                    $iw = false;
                } elseif(in_array($tn, $IW)) {
                    $new .= $ts;
                    $iw = true;
                } elseif($tn == T_CONSTANT_ENCAPSED_STRING
                    || $tn == T_ENCAPSED_AND_WHITESPACE)
                {
                    if($ts[0] == "") {
                        $ts = addcslashes($ts, "\n\t\r");
```

```
    }
    $new .= $ts;
    $iw = true;
  } elseif($tn == T_WHITESPACE) {
    $nt = @$tokens[$i+1];
    if(!$iw && (!is_string($nt) || $nt == '$') && !in_array($nt[0], $IW)) {
      $new .= " ";
    }
    $iw = false;
  } elseif($tn == T_START_HEREDOC) {
    $new .= "<<<S\n";
    $iw = false;
    $ih = true; // in HEREDOC
  } elseif($tn == T_END_HEREDOC) {
    $new .= "S;";
    $iw = true;
    $ih = false; // in HEREDOC
    for($j = $i+1; $j < $c; $j++) {
      if(is_string($tokens[$j]) && $tokens[$j] == ";") {
        $i = $j;
        break;
      } else if($tokens[$j][0] == T_CLOSE_TAG) {
        break;
      }
    }
  } elseif($tn == T_COMMENT || $tn == T_DOC_COMMENT) {
    $iw = true;
  } else {
    if(!$ih) {
      // $ts = strtolower($ts);
    }
    $new .= $ts;
    $iw = false;
  }
```

```
            }
            $ls = "";
        } else {
            if(($token != ";" && $token != ":") || $ls != $token) {
                $new .= $token;
                $ls = $token;
            }
            $iw = true;
        }
    }
    return $new;
}

?>
```

donwload.php

```
<?
        header("Pragma: public");
        header("Expires: 0");
        header("Cache-Control: must-revalidate, post-check=0, pre-check=0");
        header("Cache-Control: public");
        header("Content-Description: File Transfer");
        header("Content-type: application/octet-stream");
        header("Content-Disposition: attachment; filename=\"".
$_GET['name']."\"");
        header("Content-Transfer-Encoding: binary");
        header("Content-Length: ".filesize("./tmp/".$_GET['path']));
        readfile("./tmp/".$_GET['path']);
?>
```

ANEXO D

PROCESAMIENTO DE DATOS- NIVEL DE SATISFACCIÓN

	NIVEL DE SATISFACCIÓN DE LOS DESARROLLADORES				
1	**Que le parece el manejo del algoritmo**				
	Muy facil	3	60		60%
	Facil	2	40		40%
	Ni facil, ni complicado	0	0		
	Complicado	0	0		
	Muy complicado	0	0		
2	**El algoritmo cuenta con información acerca de su uso**				
	Totalmente de Acuerd	1	20		20%
	De acuerdo	0	0		0%
	Ni en acuerdo, Ni en d	2	40		40%
	En desacuerdo	2	40		40%
	Totalmente en Desacu	0	0		0%
3	**El algoritmo recibe múltiples archivos y carpetas empaquetadas**				
	SI	4	80		80%
	NO	1	20		20%
4	**Aplicaría el algoritmo para proteger su código fuente**				
	Totalmente de Acuerd	1	20		20%
	De acuerdo	3	60		60%
	Ni en acuerdo, Ni en d	1	20		20%
	En desacuerdo	0	0		0%
	Totalmente en Desacu	0	0		0%
5	**Usted recomendaría a otros utilizar el algoritmo**				
	Totalmente de Acuerd	1	20		20%
	De acuerdo	4	80		80%
	Ni en acuerdo, Ni en d	0	0		0%
	En desacuerdo	0	0		0%
	Totalmente en Desacu	0	0		0%
6	**Usted publicaría en los servidores los archivos ofuscados como s**				
	Totalmente de Acuerd	2	40		40%
	De acuerdo	3	60		60%
	Ni en acuerdo, Ni en d	0	0		0%
	En desacuerdo	0	0		0%
	Totalmente en Desacu	0	0		0%
					0%
7	**Como califica el algoritmo de ofuscación**				
	Muy bueno	1	20		20%
	Bueno	4	80		80%
	Regular	0	0		0%
	Malo	0	0		0%
	Muy malo	0	0		0%
8	**Al aplicar el algoritmo cual es su nivel de satisfacción**				
	Muy satisfecho	3	60		60%
	Satisfecho	2	40		40%
	Insatisfecho	0	0		0%
	Muy insatisfecho	0	0		0%
	No sirve	0	0		0%

SABANA DE DATOS

Nombres	Ciudad	P1	P2	P3	P4	P5	P6	P7	P8
Cristian Villegas Chavez	Abancay	a	d	b	b	b	b	b	b
Manuel Jesús Ibarra Cabrera	Abancay	a	d	a	b	b	a	b	a
Hesmeralada Rojas Enriquez	Abancay	a	a	a	a	a	a	a	a
Pablo Eleazar Ataucusi Romero	Andahuaylas	b	c	a	b	b	b	b	b
Elmer Quispe Peralta	Grau	b	c	a	b	b	b	b	a

UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA INFORMÁTICA Y SISTEMAS

ANEXO E

PROYECTO DE TESIS

TITULO: ALGORITMO DE OFUSCACIÓN PARA PROTECCIÓN DEL CÓDIGO FUENTE PHP COMO PROPIEDAD INTELECTUAL Y EL NIVEL DE SATISFACCIÓN DE LOS DESARROLLADORES DE SOFTWARE EN LA REGIÓN APURÍMAC – 2012.

CUESTIONARIO A LOS DESARROLLADORES DE SOFTWARE EN PHP

Nombres y apellidos:.....

NIVEL DE SATISFACCIÓN DEL PROGRAMADORES EN EL USO DEL ALGORITMO DE OFUSCACIÓN

1. **¿Qué le parece el manejo del algoritmo de ofuscación?**
   a) Muy Fácil
   b) Fácil
   c) Ni fácil ni Complicado
   d) Complicado
   e) Muy complicado

2. **El algoritmo de ofuscación cuenta con información acerca de su uso.**
   a) Totalmente de acuerdo
   b) De acuerdo
   c) Ni de acuerdo, ni en desacuerdo
   d) En desacuerdo
   e) Totalmente en desacuerdo

3. **El algoritmo de ofuscación recibe múltiples archivos y carpetas como entrada?**
   a) Si
   b) no

4. **Usted aplicaría el algoritmo para proteger su código fuente como propiedad intelectual?**
   a) Totalmente de acuerdo
   b) De acuerdo
   c) Ni de acuerdo, ni en desacuerdo
   d) En desacuerdo
   e) Totalmente en desacuerdo

5. **Usted recomendaría a otros a utilizar el algoritmo de ofuscación como protección de propiedad intelectual?**
   a) Totalmente de acuerdo
   b) De acuerdo
   c) Ni de acuerdo, ni en desacuerdo
   d) En desacuerdo
   e Totalmente en desacuerdo

6. **Usted publicaría en los servidores los archivos ofuscados como su sistema final**
   a) Totalmente de acuerdo
   b) De acuerdo
   c) Ni de acuerdo, ni en desacuerdo
   d) En desacuerdo
   e) Totalmente en desacuerdo

7. **¿Cómo calificaría el algoritmo de ofuscación ?**
   a) Muy Bueno
   b) Bueno
   c) Regular
   d) Malo
   e) Muy Malo

8. **Al aplicar el algoritmo de ofuscación, ¿Cuál es su nivel de satisfacción?**
   a) Muy satisfecho
   b) Satisfecho
   c) Insatisfecho
   d) Muy insatisfecho
   e) No sirve