Universidad de las Ciencias Informáticas

FACULTAD #5

ENTORNOS VIRTUALES

Título: Aseguramiento de Recursos Multimedia para Sistemas de Realidad Virtual.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Tatiana Cervantes del Toro

Sanjony Yasmany O'Reilly Lebrijio

Tutor: Ing. Orlay García Duconge

DECLARACIÓN DE AUTORÍA

Declaramos ser únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de _____ del año _____.

Tatiana Cervantes del Toro

_____

Firma del Autor

Sanjony Yasmany O'Reilly Lebrijio

_____

Firma del Autor

Ing. Orlay García Duconge

_____

Firma del Tutor

DATOS DEL CONTACTO

Nombre y Apellidos: Orlay García Duconge.

Edad: 25 Años

Ciudadanía: cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Categoría Docente: Instructor Recién Graduado.

E-mail: oducunge@uci.cu.

Graduado en Ingeniería en Ciencias Informáticas, en la Universidad de las Ciencias Informáticas (UCI), actualmente profesor de esta misma escuela, además de ser el Arquitecto del Dominio Visualización 3D del Polo de Realidad Virtual de la Facultad 5.

AGRADECIMIENTOS

Numerosas son las personas que han contribuido a la realización de esta investigación, no sólo aportando sus conocimientos, sino brindándome cariño y mucho amor.

Primeramente quiero agradecer al dúo más importante en mi vida, mis padres.

A ti mami, por tu apoyo incondicional desde el primer momento en que decidí estudiar en esta Universidad, por tu amor sin límites.

A ti papi, por educarme desde pequeña con esa rectitud que te caracteriza, eso es lo que me impulsó a ser de mi quien soy.

A mí querida abuelita, por siempre darme ánimos en los momentos difíciles y apoyarme sin fronteras con sus ideas y también por su amor sin límites.

A mí querido abuelito, que aunque ya haya fallecido siempre quiso que yo me hiciera alguien en la vida y mientras estuvo conmigo siempre tuve su apoyo y su cariño incondicional en todo momento.

A mí querida tía Blanqui, por velar por mí día y noche desde que llegue aquí a La Habana, por haber sido siempre mi segunda madre, has cumplido tu promesa.

A mí Bebi, a quien quiero tanto por soportar de vez en cuando mis pesadeces, por siempre estar a mi lado brindándome su cariño y apoyándome en todo momento.

A mis otras tías Noni, Margarita, Isa a todas ellas por siempre haber estado al tanto de mis estudios y también todas me brindaron mucho cariño que yo considero que fue muy importante.

A mí tío Mario, por haber ejercido la función de padre en todo el tiempo que estuve aquí con mucho amor.

A todos mis compañeros que de una forma u otra aportaron su granito de arena en mi investigación y también a todos los que estuvieron conmigo durante mi estancia en la universidad, pero en especial quiero agradecerle a mi compañero de tesis Sanjony por haber aguantado mi impertinencia durante el desarrollo de la misma.

A mí tutor, por su apoyo brindado.

A todos muchas gracias por su cariño y amor sin límites.

TATIANA

Agradecimientos

Incontables son las personas que han hecho posible que haya llegado al lugar donde estoy, por lo que siempre estaré eternamente agradecido por el empeño y confianza que depositaron en mí.

Especiales agradecimientos:

Primeramente a mi madre, que ha sido madre y padre a la vez, por su apoyo y entrega desde el mismo momento en que empecé esta carrera y durante toda mi vida.

A mis tías, que siempre han estado pendientes de mi recorrido universitario.

A mi abuela, que siempre ha tenido mente positiva sobre mi desenvolvimiento en la carrera.

A mi hermanita, que aunque no tenga ningún conocimiento sobre el contenido de mi carrera, me brinda su amor y preocupación incondicional.

A mis amigos y vecinos del barrio, que siempre han confiado en poder contar con un ingeniero informático.

A todos mis compañeros, que han transitado conmigo momentos difíciles y momentos alegres que nunca se olvidarán a lo largo de toda la trayectoria universitaria.

A los amigos especiales que siempre me apoyan (Héctor, Alexis, Pavel, Jose, Darluin y Yandry), por compartir y ser piezas de apoyo en cualquier circunstancia.

A mi tutor, por su apoyo brindado.

En fin, a todos muchas gracias por de una forma u otra, aportar su granito de arena en esta investigación y en la formación de un ingeniero informático.

SANJONY

DEDICATORIA

A mis dos pilares de la educación, a mi mamá y a mi papá a ellos les debo lo que soy.

A mis hermanos para que sigan este camino.

TATIANA

A mi familia que ha depositado toda su confianza en mí, para poder formarme como ingeniero informático.

SANJONY

RESUMEN

Este trabajo propone una aplicación para proteger los recursos empleados en aplicaciones gráficas desarrolladas sobre el motor gráfico OGRE. Surge a partir de la necesidad de evitar la piratería de recursos.

El trabajo se centra en el aseguramiento de recursos multimedia para Sistemas de Realidad Virtual. Se propone como variante de solución, el empleo de funciones de la biblioteca “dirent” para el manejo de ficheros sobre ambas plataformas (Windows y Linux). Además se propone el uso de un algoritmo criptográfico simétrico llamado Rijndael de un alto rango de seguridad para el procesamiento binario de los ficheros a encriptar.

El presente trabajo propone obtener una aplicación que brinda las funcionalidades para encriptar todo tipo de recurso pero solo nos enmarcaremos en los siguientes: texturas, modelos 3D y scripts.

PALABRAS CLAVES

Encriptación
Desencriptación
Fichero
Rijndael
Criptografía
Cifrado
Algoritmo

Tabla de Contenido

Tabla de Contenido

INTRODUCCIÓN

En el transcurso de los años, la Informática ha obtenido avances relevantes en la implementación de software. El auge de la tecnología ha permitido un positivo desenvolvimiento en la esfera de las aplicaciones gráficas como proceso de diseño de modelos en tiempo real, que permiten comprender el comportamiento del sistema evaluando nuevas estrategias y criterios. En el desarrollo de proyectos de simulación están vigentes elementos y recursos que son vitales y constituyen un eslabón fundamental en el buen funcionamiento del producto. La protección y aseguramiento de estos componentes hacen más fiables y auténticos los productos en cuestión.

En Cuba, se encuentran varios centros que se han dedicado a impulsar el desarrollo de la industria del software, dando alcance hoy día a notables logros en el marco internacional.
Para la informatización de nuestro país se crea la Universidad de las Ciencias Informáticas (UCI) en el 2002, con el objetivo de insertar al país en los principales escalones de la producción mundial de software.

La UCI, enmarcada en la rama de producción, crea el perfil de Entornos Virtuales, dentro del cual, los proyectos existentes cuentan con varios módulos para la implementación de software de buena calidad que puedan emerger en el mercado mundial.

Desarrollar una aplicación de software implica una gran inversión de tiempo, dinero y esfuerzo. La protección de los recursos con los que se desarrollan los software es una parte importante del desarrollo informático. Con la revolución digital se ha generado un crecimiento en la piratería, que está cambiando de forma irreversible la manera de desarrollar aplicaciones. La piratería informática reprime y frena la innovación. El coste de combatir la piratería informática, junto con la reducción de ingresos, podría invertirse en investigación y desarrollo para beneficiar a los usuarios; además es un obstáculo al crecimiento de la industria de software, y acarrea un verdadero atraso en el desarrollo de mejores aplicaciones para todos.

Los productos que se obtienen en el marco de desarrollo de la facultad 5, cuentan con la utilización de importantes recursos tales como: imágenes, modelos 3D, ficheros de sonido y ficheros de video, scripts y otros elementos indispensables, pero no disponen de un sistema capaz de darle una protección consistente a estos componentes esenciales; lo que da como resultado un amplio marco de vulnerabilidad de los productos que hoy se desarrollan.

Analizando detenidamente la problemática anterior, este trabajo propone como **problema científico** a resolver: ¿Cómo proteger y asegurar los recursos multimedia de las aplicaciones gráficas que se desarrollan?

De esta forma, se plantea como **objeto de investigación**, los procesos para la seguridad de los recursos.

Se tendrá como **Objetivo General de Investigación**:
Desarrollar un método para asegurar recursos de nuestras aplicaciones gráficas.

Nos enfocaremos en el **campo de acción**, encriptación y desencriptación de los recursos empleados en el desarrollo e implementación de aplicaciones gráficas.

Para el cumplimiento de los objetivos planteados anteriormente se trazan las siguientes **tareas a desarrollar**:

1- Identificación de los métodos que se utilizan actualmente en el aseguramiento de los recursos multimedia para seleccionar cual será empleado.
2- Selección de herramientas compatibles con el método seleccionado, para desarrollar el sistema.
3- Enumeración de las ventajas y desventajas del método seleccionado.
4- Diseño de la solución.
5- Implementación de la solución.

En sentido general, la confección de la aplicación de encriptación y desencriptación brindará un mecanismo de protección para los recursos multimedia que se utilizan en las aplicaciones gráficas que se desarrollan en nuestro centro.

CAPÍTULO #1. FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se relacionan los aspectos más importantes de la historia de los algoritmos de criptografía, principales características que presentan los algoritmos de criptografía así como mencionar los algoritmos de criptografía más utilizados, obtenidos a través de un estudio de la bibliografía existente sobre el tema.

1.1- Marco Teórico

La delincuencia informática parece que va más rápida que su seguridad. Con la expansión de la red se ha acelerado el desarrollo de las técnicas de protección de aplicaciones gráficas en grandes instituciones. El desarrollo de las telecomunicaciones en estos últimos años ha creado toda una variedad de nuevas necesidades. La seguridad de los sistemas informáticos se ve debilitada por el fuerte crecimiento de las redes. El reciente aumento del uso de las tecnologías han dirigido la atención del mundo entero a un problema crucial: la **privacidad** y la **integridad de datos**.

Si tenemos en cuenta la cantidad de datos almacenados en bases de datos informatizadas o la utilización cada vez mayor del correo electrónico, se comprende la necesidad de encontrar sistemas lo más seguros posible que garanticen la confidencialidad de datos y comunicaciones.

Con la expansión de la red se ha acelerado el desarrollo de las técnicas de ocultación, ya que, al mismo ritmo que crece la libertad de comunicarse, se multiplican los riesgos para la privacidad. Algunas empresas ya han estado tomando medidas para ofrecer al usuario un elevado nivel de seguridad en lo que se refiere al almacenamiento y protección de recursos media, donde resalta una técnica para proteger la confidencialidad: el **cifrado**. **[1]**

Los campos de aplicación de la criptografía son tan amplios como la imaginación nos permita. En cualquier circunstancia se encuentran situaciones en las que sería necesario un nivel de seguridad más alto. Por ejemplo, el acceso a información privada en nuestro entorno particular, o corporativa en nuestro entorno laboral; operaciones monetarias, desde comprar con nuestra tarjeta de crédito en un comercio tradicional, hasta ordenar una transferencia en un banco por Internet. En los entornos de negocio, se abren nuevas posibilidades de utilización de métodos criptográficos. Uno de los ejemplos de este tipo de entorno es:

El motor gráfico OGRE (Object-Oriented Graphics Rendering Engine) no dispone actualmente de ningún sistema para la protección de sus recursos, por lo que se están estudiando diversos métodos para suprimir dicho problema y mantener de una forma segura sus recursos. Para ello, plantean varias técnicas como son: el cambio de las extensiones a los archivos convirtiéndolos en formatos desconocidos, para así, desviar la atención de intrusos que quieran acceder a los recursos propios del sistema. Además, se

propone la utilización de sistemas de cifrado tales como: cifrado XOR, cifrado digital, por sus grandes características y ventajas. **[2]**

Cifrado XOR: consiste en comparar un bit del mensaje con el bit correspondiente de la clave. Si esos bits son diferentes, en el texto cifrado se pone un (1) en el mismo sitio. Si son iguales, se pone un (0) en el mensaje codificado. Con esta misma operación puedes recuperar también el texto original.

Cifrado Digital: dado un mensaje en claro, es decir, mensaje reconocible, al que se le aplique un algoritmo de cifrado, se generará como resultado un mensaje cifrado que sólo podrá ser descifrado por aquellos que conozcan el algoritmo utilizado y la clave que se ha empleado.

1.2-Criptografía

La criptografía (del griego κρύπτω krypto, «oculto», y γράφω graphos, «escribir», literalmente «escritura oculta») es el arte o ciencia de cifrar y descifrar información mediante técnicas especiales y es empleada frecuentemente para permitir un intercambio de mensajes que sólo puedan ser leídos por personas a las que van dirigidos y que poseen los medios para descifrarlos.

Con más precisión, cuando se habla de esta área de conocimiento como ciencia se debería hablar de criptología, que a su vez engloba tanto las técnicas de cifrado, es decir la criptografía propiamente dicha, como sus técnicas complementarias, entre las cuales se incluye el criptoanálisis, que estudia métodos empleados para romper textos cifrados con objeto de recuperar la información original en ausencia de las claves. **[3]**

1.3-Historia de la criptografía

La historia de la criptografía se remonta a miles de años. Hasta décadas recientes, ha sido la historia de la criptografía clásica — los métodos de cifrado que usan papel y lápiz, o quizás ayuda mecánica sencilla. A principios del siglo XX, la invención de máquinas mecánicas y electromecánicas complejas, como la máquina de rotores Enigma, proporcionaron métodos de cifrado más sofisticados y eficientes; y la posterior introducción de la electrónica y la computación ha permitido sistemas elaborados que siguen teniendo gran complejidad.

La evolución de la criptografía ha ido de la mano de la evolución del criptoanálisis — el arte de "romper" los códigos y los cifrados. Al principio, el descubrimiento y aplicación del análisis de frecuencias a la lectura de las comunicaciones cifradas ha cambiado en ocasiones el curso de la historia. De esta manera, el telegrama Zimmermann provocó que Estados Unidos entrara en la Primera Guerra Mundial; y la lectura, por parte de los Aliados, de los mensajes cifrados de la Alemania nazi, puede haber acortado la Segunda Guerra Mundial hasta dos años.

Hasta los años 70, la criptografía segura era dominio casi exclusivo de los gobiernos. Desde entonces, dos sucesos la han colocado de lleno en el dominio público: la creación de un estándar de cifrado público (DES); y la invención de la criptografía asimétrica. **[4]**

Técnicas que usa la criptografía

- Sustitución
- Trasposición
- Checksums, hashing o variantes de éstas

La **sustitución** reemplaza un carácter o conjunto de caracteres por otro conjunto de caracteres. Matemáticamente la podemos expresar como una función que asigna a un valor x, un valor y.

Sust: N(L) → N(L+A)

Esto quiere decir que la longitud de los datos de salida (imagen) es la longitud de los de entrada más/menos un valor entero positivo A. En general, A ≈ 0. Si A < 0, estamos hablando de compresión (que en cierto sentido también es encriptación) y el mensaje encriptado reduce su longitud. Si A = 0 es probable que el mensaje se halla codificado reemplazando cada carácter por otro, posiblemente bajo un sistema criptográfico ROT.

$$\text{Sust: } f(x1\ x2...) = a1\ a2...$$

$$(\text{con } x1\ x2... \neq a1\ a2..., \text{long}(x1) \geq 0, \text{long}(x2) \geq 0, \text{long}(xi) \geq 0, \text{long}(x1) + \text{long}(x2) +... \geq 1)$$

La **trasposición** cambia el orden de los caracteres o conjuntos de caracteres. Usando esta técnica la longitud del mensaje encriptado es igual a la del mensaje original.

Los **checksums**, por su parte, son valores numéricos que se obtienen de sumar todos los valores de los caracteres del mensaje. Luego de hacer esta suma, su valor se agrega al final del mensaje. Cuando se procesa el mensaje se hace nuevamente esta suma y tiene que ser igual a la anterior que está almacenada en el mensaje. Esto sirve para saber si el mensaje fue modificado ilegalmente (por ej. con un editor de texto). Para poder modificarlo y evitar que sea detectada esta acción deberíamos cambiar no sólo el mensaje sino también la suma de verificación.

Extendiendo la idea de checksum, se pueden generar otros valores computados a partir del mensaje usando hashing. Esto daría una especie de resumen que nos firma el mensaje de acuerdo a su contenido.

Hay que nombrar que esta técnica es muy usada en las firmas digitales, de modo que se consigue una clave privada que relaciona la clave pública con el contenido del mensaje. Métodos similares se usan en los archivos comprimidos para saber si el mensaje se preserva en estado consistente; y hasta en algunos dispositivos de almacenamiento como discos rígidos (a muy bajo nivel, implementado en hardware).

La sustitución en general se basa en problemas matemáticos complejos, muchas veces relacionados con la teoría de números: números primos, álgebra de módulo, divisiones enteras, etc. Estos problemas son ideales para encriptar ya que la operación de desencriptado consiste en resolverlos, y esto puede tomar tiempos exponenciales. En general se usan problemas cuya solución rápida o analítica se desconoce, y la única manera de obtenerla es probando una por una.

Resolver un problema como la factorización de un número muy grande en sus factores primos hoy en día es complicado. No se han descubierto algoritmos eficientes de factorización, lo que significa que debemos usar la fuerza bruta para romper claves. Usar fuerza bruta es probar con todas las posibilidades, hasta que se encuentre la correcta. Por ejemplo: si tenemos una clave (contraseña) de 32 bits (4 bytes o 4 caracteres ASCII) para probar todas las posibilidades tendremos que evaluar 2^32 claves. Esto ocurre si la correcta es justo la última. Entonces es mejor calcular la media (promedio) del total de pruebas. 2^32/2=3^31. Sigue siendo un número grande, más de 2000 millones.

¿Por qué es 2^n?
Imagen que tenemos 4 bits, donde cada uno puede tomar un valor 0 o 1
Para 1 bit tenemos 2 combinaciones: 0 o 1.
Para 2 bits tenemos 4 combinaciones: 00, 01, 10, 11.

Para 3 bits tenemos 8 combinaciones: 000, 001, 010, 011, 100, 101, 110, 111.
Para 4 bits tenemos 16 combinaciones.

Ahora haremos una clasificación de acuerdo a la variación de los códigos utilizados.

- **sustitución simple:** cada carácter se reemplaza por otro. Por ej. a -> n. La correspondencia no cambia en todo el cifrado. Un ejemplo es el sistema criptográfico ROT. Es el cifrado más inseguro, y posibilita la detección de repeticiones de caracteres, como espacios por ej.
- **sustitución en intervalos**: los caracteres se reemplazan por otros pero se repiten cada determinado intervalo.
  Por ej. a→n, b→x,..... , a→w, b→x, ... a→n,
  b→x. Aquí vemos que la primera "a" se sustituye por la letra "n", la segunda "a" por la letra "w". Pero después de cierto intervalo, la "a" se reemplaza nuevamente por "n".
- **sustitución azarosa:** cada carácter se reemplaza por uno distinto, y no hay una ley precisa que determine los intervalos de repetición. Por ej. a→n, b→x,.....
  a→w, b→x,..... a→1, b→2,...

Es decir que cada carácter nunca se va a reemplazar por un mismo carácter con igual frecuencia. Es la generalización de las 2 primeras clasificaciones, siendo la más segura de todas. Además no permite detectar caracteres repetidos. Una desventaja de ésta es que es muy probable que deje al mensaje con poca capacidad de compresión, o sea que casi no lo podremos comprimir por los métodos tradicionales basados en redundancia de información.

Un ejemplo es el software Seyo 2 que tiene implementados los 3 niveles. El algoritmo básico ByteXByte es el caso de sustitución simple. El algoritmo xorizador nos provee de sustitución en intervalos. Y el más seguro de todos es motorikke, que brinda sustitución azarosa. El problema de la sustitución presentado hasta aquí parece ser trivial, pero no lo es. Para desarrollar un sistema criptográfico sus creadores parten de la base de que el algoritmo de cifrado es público, se conoce. Es posible obtenerlo mediante técnicas de ingeniería inversa. Por eso el secreto no debe estar en el algoritmo, sino en la clave. Un buen algoritmo criptográfico es aquel que depende de su clave. Dependiendo de su valor, el algoritmo se comportará de una u otra manera. Así tenemos millones de algoritmos posibles que se pueden generar a partir de todas

las claves posibles. Como ven, no es algo trivial; menos si se trata de cifrado asimétrico (con 2 claves, una para encriptar y otra para desencriptar). **[10]**

1.4-Tipos de Algoritmos de Criptografía

Un método de encriptación y desencriptación se llama un cifre. Algunos métodos de la criptografía confían en el secreto de los algoritmos; los algoritmos sólo son de interés histórico y no son adecuados para necesidades del mundo real. Todos los algoritmos modernos usan una llave para controlar encriptación y desencriptación; un mensaje sólo puede ser desencriptado si la llave de desencriptar empareja la llave de encriptar.

Hay dos clases de algoritmos de encriptación usando llaves, los algoritmos **simétricos** (o llave-secreto) y **asimétricos** (o llave-público). La diferencia es que los algoritmos simétricos usan la misma llave para encriptar y desencriptar (o la llave del desencriptar se deriva fácilmente de la llave de encriptar), considerando que los algoritmos asimétricos usan una llave diferente para encriptar y desencriptar, y la llave de desencriptar no puede derivarse de la llave de encriptar. **[5][6]**

1.5-Algoritmos Criptográficos

1.5.1- Algoritmos Simétricos.

Los algoritmos simétricos más conocidos son:

DES (Data Encryption Standard): Utiliza bloques de 64 bits, los cuales codifica empleando claves de 56 bits y aplicando permutaciones a nivel de bit en diferentes momentos (mediante tablas de permutaciones y operaciones XOR). Es una red de Feistel de 16 rondas, más dos permutaciones, una que se aplica al principio y otra al final. La flexibilidad de DES reside en que el mismo algoritmo puede ser utilizado tanto para cifrar como para descifrar, simplemente invirtiendo el orden de las 16 subclaves obtenidas a partir de la clave de cifrado. En la actualidad no se ha podido romper el sistema DES criptoanalíticamente (deducir la clave simétrica a partir de la información interceptada).

Sin embargo una empresa española sin fines de lucro llamado Electronic Frontier Foundation (EFF) construyo en Enero de 1999 una máquina capaz de probar las 256 claves posibles en DES y romperlo sólo en tres días con fuerza bruta. A pesar de su caída DES sigue siendo utilizado por su amplia extensión de las implementaciones vía hardware existentes (en cajeros automáticos y señales de video por ejemplo)

y se evita tener que confiar en nuevas tecnologías no probadas. En vez de abandonar su utilización se prefiere suplantar a DES con lo que se conoce como cifrado múltiple, es decir aplicar varias veces el mismo algoritmo para fortalecer la longitud de la clave. **[5][6][8][23]**

DESX: DES-X (o DESX) es una variante del DES (Data Encryption Standard) de cifrado de bloques destinados a aumentar la complejidad de un ataque de fuerza bruta usando una técnica denominada blanqueamiento clave.

El algoritmo DES original se especifica en el 1976 con unos 56-tamaño bits clave: 2 posibilidades para la clave. Hubo críticas de que una búsqueda exhaustiva podría ser dentro de las capacidades de los gobiernos grandes, en particular los Estados Unidos "National Security Agency (NSA). Un plan para aumentar el tamaño de la clave de DES sin alterar sustancialmente el algoritmo DES-X, propuesta por Ron Rivest mayo 1984.

El algoritmo se ha incluido en RSA Security 's biblioteca criptográfica BSAFE desde finales de 1980.

DES-X aumenta DES por XORing un extra de 64 bits de la clave (K1) al texto en claro antes de la aplicación de DES y, a continuación otra operación XOR de 64 bits de la clave (K2) después de cifrado:

$$\text{DES-X}(M)=K_2 \oplus \mathrm{DES}_k(M \oplus K_1)$$

El tamaño de la clave es lo que aumentó a 56 + 2 × 64 = 184 bits. Sin embargo, el tamaño de la clave efectiva (de seguridad) sólo aumentó a 56 +64-1- libras (M) = 119 - libras (M) = ~ 119 bits, en donde M es el número de texto conocido pares de texto cifrado que el adversario la posibilidad de obtener, y la libra representa el logaritmo en base. (Debido a esto, algunas implementaciones de K2 realmente hacen una función de una manera fuerte de K1 y K).

DES-X también aumenta la fuerza de la DES contra el criptoanálisis diferencial y el criptoanálisis lineal, aunque la mejora es mucho menor que en el caso de los ataques de fuerza bruta. Se estima que el criptoanálisis diferencial requiere 2 textos planos escogidos (vs 2 de DES), mientras que el criptoanálisis lineal requeriría 2 textos planos conocidos (frente a 2 de DES.) Tenga en cuenta que con 2 textos planos (conocido o ser elegido el mismo en este caso), DES (o cualquier cifrado en bloque de

otros 64 con un tamaño de bits bloque) es totalmente rota por el ataque de libro de códigos elementales. **[23]**

Triple-DES (T-DES): Triple DES es una variante menor del estándar DES. Toma una llave de 192 bit (24 caracteres) como entrada y lo divide en tres llaves. Primero, DES es usado para codificar un archivo usando la primera llave. Luego el archivo es decodificado utilizando la segunda llave. La etapa final es codificar nuevamente el archivo usando la tercera llave. Note que si las tres llaves de 64 bit son iguales, Triple DES es idéntico a DES simple. Sin embargo, si este método es utilizado correctamente, la codificación es mucho más segura que con DES simple.

Consiste en aplicar 3 veces DES de la siguiente manera:

1. Se codifica con la clave K^1.
2. Se decodifica el resultado con la clave K^2.
3. Lo obtenido se vuelve a codificar con K^1.

La clave resultante es la concatenación de K^1 y K^2 con una longitud de 112 bits. **[5][6][23]**

IDEA (International Data Encryption Algorithm): Es un algoritmo desarrollado en ETH Zurich en Suiza por Xuejia Lai. Usa una llave de 128 bits y generalmente se considera muy seguro. Ha sido uno de los mejores algoritmos públicos conocidos por un tiempo (antes de que el estándar AES comenzara su segunda ronda, ver arriba). Ha circulado varios años y no se han publicado ataques prácticos a pesar de numerosos intentos de analizarlo.

Uno de los mejores ataques usa la imposible idea diferencial de Biham, Shamir y Biryukov. Pueden atacar solo 4.5 redondeos de IDEA, lo que no supone riesgos sobre el total de 8.5 redondeos usados en IDEA.

El proceso de encriptación consiste ocho rondas de cifrado idéntico, excepto por las subclaves utilizadas (segmentos de 16 bits de los 128 de la clave), en donde se combinan diferentes operaciones matemáticas (XORs y Sumas Módulo 16) y una transformación final. **[5][6]**

BlowFish: Blowfish fue diseñado en 1993 por Bruce Schneier. Desde entonces ha sido utilizado considerablemente y aceptado como un algoritmo de codificación fuerte.
Para la encriptación emplea bloques de 64 bits y permite claves de encriptación de diversas longitudes (hasta 448 bits).

Generalmente, utiliza valores decimales de Π (aunque puede cambiarse a voluntad) para obtener las funciones de encriptación y desencriptación. Estas funciones emplean operaciones lógicas simples y presentes en cualquier procesador. Esto se traduce en un algoritmo "liviano", que permite su implementación, vía hardware, en cualquier controlador (como teléfonos celulares por ejemplo).

Los únicos ataques conocidos contra Blowfish se basan en sus pobres clases de llaves. **[5][6][23]**

RC5: Este algoritmo, diseñado por RSA (4), permite definir el tamaño del bloque a encriptar, el tamaño de la clave utilizada y el número de fases de encriptación. El algoritmo genera una tabla de encriptación y luego procede a encriptar o desencriptar los datos. **[5][6]**

CAST: Es un buen sistema de cifrado en bloques con una clave CAST-128 bits, es muy rápido y es gratuito. Su nombre deriva de las iniciales de sus autores, Carlisle, Adams, Stafford Tavares, de la empresa Northern Telecom (NorTel). CAST no tiene claves débiles o semidébiles y hay fuertes argumentos acerca que CAST es completamente inmune a los métodos de criptoanálisis más potentes conocidos. También existe una versión con clave CAST-256 bits que ha sido candidato a AES. **[5][6]**

MARS: Es un bloque cipher de llave compartida (simétrico) diseñado por IBM como un algoritmo candidato para AES. MARS admite bloques de 128-bits y un tamaño variable de llave.

Fue seleccionado como uno de los cinco finalistas en el concurso de AES. MARS es único en cuanto combina prácticamente cualquier diseño técnico conocido por los codificadores en un solo paquete. Utiliza dos algoritmos completamente separados, de manera tal que si uno de ellos ha sido quebrado, el resto del cipher permanecerá seguro así como la información.

Debido a su diseño, MARS ofrece una mejor seguridad que Triple DES, y al mismo tiempo, tiene una velocidad de ejecución significativamente mayor que Single DES. **[5][6]**

SERPENT: Serpent tiene un diseño básicamente conservador pero en muchos aspectos innovador.

Serpent es un algoritmo de cifrado simétrico de bloques que quedó finalista en el concurso Advanced Encryption Standard del NIST, tras Rijndael.

Serpent fue diseñado por Ross Anderson, Eli Biham y Lars Knudsen.

Como otros participantes del AES Serpent usa un tamaño de bloque de 128 bits y soporta tamaños de clave de 128, 192 y 256 bits de longitud. El cifrado consiste en 32 rondas de substitución-permutación operando sobre cuatro bloques de 32 bits. Cada ronda usa 32 copias de la misma S-Box de 4-bit a 4-bit. Serpent se diseñó para que las operaciones se realizasen en paralelo, usando 32 desplazamientos de 1 bit.

Serpent adoptó una visión mucho más cauta que otros participantes al AES, optando por un mayor margen de seguridad. Los diseñadores afirmaron que 16 rondas serían suficientes para lo métodos conocidos de ataque, pero especificaron 32 rondas para asegurarse de la robustez del algoritmo contra futuros descubrimientos en criptoanálisis. **[5][6]**

TWOFISH: Es un nuevo cifrado en bloque diseñado por Counterpane (cuyo fundador y CTO es Bruce Schenier). El diseño es altamente delicado, con muchas maneras alternativas de implementación. Es criptoanalizado en detalle, por el autorizado "equipo extendido de TwoFish". Es básicamente un cifrado Feistel, pero utiliza varias ideas diferentes.

El tamaño de bloque en Twofish es de 128 bits y el tamaño de clave puede llegar hasta 256 bits. Twofish llegó a la ronda final del concurso del NIST, pero no fue elegido para la estandarización. TwoFish quedó tercero, tras Rijndael y Serpent.

Twofish fue diseñado por Bruce Schneier, John Kelsey, Doug Withing, David Wagner, Chris Hall y Niels Ferguson. El Extended Twofish team, que realizó más profundos criptoanálisis de Twofish y otros participantes a AES incluyen a Stefan Lucks, Tadayoshi Kohno, y Mike Stay.

Twofish se relaciona con el método de cifrado por bloques anterior Blowfish. Las características distintivas de Twofish son el uso de S-boxes pre-computadas con llaves dependientes, y una llave-horario relativamente compleja. Twofish coge prestados algunos elementos de otros diseños: por ejemplo, el Pseudo-Hadamard transforman (PHT) de la familia SAFER de cifrado. Twofish utiliza la misma estructura de Feistel que el DES.

En la mayoría de las plataformas de software Twofish es levemente más lento que Rijndael (el algoritmo elegido para AES) para las llaves de 128 bits, pero algo más rápido para las llaves de 256 bits.

Este cifrado tiene S-boxes dependientes de llaves como Blowfish (otro cifrado de Bruce Schneier). **[23]**

3WAY: Es un cifrado de bloques diseñado en 1994 por Joan Daemen, que también (con Vincent Rijmen) diseñado Rijndael, el ganador del NIST 's concurso de Advanced Encryption Standard (AES).

3Way tiene un tamaño de bloque de 96 bits, en particular, no una potencia de dos, como el más común 64 o 128 bits. La longitud de la clave es 96 bits. La figura 96 se deriva de la utilización de palabras de tres 32 bits en el algoritmo, de la que también se deriva el sistema de cifrado nombre. 3-Way, cuando se inventó, 96-bit y bloques de teclas son bastante fuertes, pero cifras más recientes tienen un bloque de 128-bits, y ahora tienen menos de claves de 128 bits. 3-Way es un 11-la sustitución de ronda red permutación.

3Way está diseñado para ser muy eficaz en una amplia gama de plataformas de 8-bits para procesadores de hardware especializado, que tiene algunas características que permiten elegante matemática casi todo el descifrado por hacer exactamente en los mismos circuitos al igual que el de cifrado.

3Way es vulnerable al criptoanálisis clave relacionadas, John Kelsey, Bruce Schneier, y David Wagner demostrar cómo puede romperse con una consulta clave relacionadas y sobre 2 textos planos escogidos. **[23]**

Rijndael (El nuevo estándar AES):

Conceptos Matemáticos preliminares

El algoritmo Rijndael opera a nivel de bytes, interpretando estos como elementos de un cuerpo de Galois GF (2^8), y a nivel de registros de 32 bits, considerándolos como polinomios de grados menor de 4 con coeficientes que son a su vez polinomios en GF (2^8). Aquí se van a definir las operaciones matemáticas básicas que necesita el algoritmo Rijndael así como algunos conceptos referentes al tratamiento de polinomios.

Cuerpos Finitos.GF (2^8)

En este algoritmo todos los bytes se interpretan como elementos de un cuerpo finito. Concretamente, se representan mediante Campos de Galois que se representan como GF(k) (”Galois Field”).

Los campos de Galois son muy interesantes en criptografía, gracias a que existe un inverso aditivo y multiplicativo que permite cifrar y descifrar en el mismo cuerpo $Z_{k,}$ eliminado así los problemas de redondeo o truncamiento de valores si tales operaciones de cifrado y descifrado se hubiesen realizado en aritmética real.

En nuestro caso, interesa utilizar una aritmética en módulo "p" sobre polinomios de grado "m", siendo "p" un número primo. Este campo de Galois queda representado como: GF (p^m), en donde los elementos de GF (p^m) se representan como polinomios con coeficientes Z_p de grado menos que m, es decir:

$$GF(p^m)=\{\lambda_0 + \lambda_1 x + \lambda_2 x^2 + \dots \lambda_{m-1} x^{m+1};\ \lambda_{0,}\ \lambda_1,\ \lambda_{2\dots}\ \lambda_{m-1} \in Z_p\}$$

Cada elemento de GF (p^m) es un resto módulo p(x), donde p(x) es un polinomio irreducible de grado "m", esto es, que no puede ser factorizado en polinomios de grado menor que "m".

En el caso del algoritmo Rijndael, será interesante los campos del tipo GF(2^m) puesto que los coeficientes en este caso serán los restos del módulo 2, es decir, 0 y 1 , lo que permite una representación binaria. Por lo tanto, cada elemento del campo se representa con m bits y el número de elementos será 2^m.

Por ejemplo, para el campo GF(2^3) sus elementos son:
0, 1, x, x+1, x^2 +i, x^2 +x, x^2 +x +1 que son precisamente todos los restos de un polinomio de grado "m-1".

En el caso del algoritmo Rijndael, se definen operaciones a nivel de byte, encontrándonos en el campo GF(2^8).

Un byte "B", se compone de los bits b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0, si lo consideramos como un polinomio con coeficientes en {0,1} tenemos el polinomio:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Por ejemplo, un byte con el valor hexadecimal "57", en binario 01010111, corresponde con el polinomio: $x^6 + x^4 + x^2 + x + 1$.

Suma en GF(2^8)

En GF(p^m) hay que considerar que las operaciones matemáticas sobre los coeficientes se hacen en módulo "p" con lo cual en GF(2^m) se reducen los resultados de la suma de los coeficientes módulo 2. Este procedimiento tanto para la suma como para la resta se realiza simplemente con una operación Or-Exclusiva, ya que si los coeficientes son iguales darán como suma o resta un 0 y coeficientes distintos darán un 1.

Por tanto, para sumar dos polinomios, basta con aplicar la operación Or-Exclusiva, a cada elemento de los polinomios, dos a dos. La función Or-Exclusiva de dos bits, produce los siguientes resultados:

$$1 \oplus 1 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Un ejemplo de suma de dos polinomios de tamaño byte expresados dentro de GF(2^8) es:

$$A = 57_{i6} = 0101\ 0111_2 = x^6 + x^4 + x^2 + x + 1$$
$$B = 83_{i6} = 1000\ 0011_2 = x^7 + x + 1$$

Sumando:

$$A+B = (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) \bmod 2$$
$$A+B = (x^7 + x^6 + x^4 + x^2 + 2x + 2) \bmod 2 = x^7 + x^6 + x^4 + x^2 =$$
$$= 1101\ 0100 = D4_{16}$$

Y lo mismo se obtiene utilizando la operación Or-Exclusiva:

$$0101\ 0111 \oplus 1000\ 0011 = 1101\ 0100 = D4_{16}$$
$$\{57\} \oplus \{83\} = \{D4\}$$

Multiplicación en GF(2^8)

En la multiplicación de polinomios en GF(2^m), es posible que el resultado contenga elementos que estén fuera del cuerpo del polinomio (potencias iguales o mayores que "m") por lo que deberemos reducir los exponentes mediante un polinomio p(x) necesariamente irreducible y grado "m".

Para GF(2^8) la multiplicación de polinomios se realiza modulo con un polinomio irreducible de grado 8. Este polinomio irreducible se representa por m(x) (es irreducible porque sus únicos divisores son el 1 y el mismo polinomio). El polinomio irreducible utilizado en el algoritmo Rijndael es:

$$m(x) = x^8+x^4+x^3+x+1$$

Ejemplo de una multiplicación de polinomios: '57' • '83' = 'C1', esto es así porque:

$$A = 57_{16} = 0101\ 0111_2 = x^6 + x^4 + x^2 + x + 1$$
$$B = 83_{16} = 1000\ 0011_2 = x^7 + x + 1$$

$A \bullet B = (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + 2x^2 + 2x + 1 \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$X^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod x^8 + x^4 + x^3 + x + 1 = x^7 + x^6 + 1 = = 1100\ 0001 = C1_{16}$

Otra forma de calcularlo sería razonando qué resultado de la multiplicación hay que reducirlo por m(x) para cada valor de x que esté fuera del cuerpo de 8 bits:

Sea $m(x) = x^8 + x^4 + x^3 + x + 1 \Rightarrow x^8 = x^4 + x^3 + x + 1$

$A \bullet B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$X^{13} = x^5 * x^8 = x^5 * (x^4 + x^3 + x + 1) = x^9 + x^8 + x^6 + x^5 = x * x^8 + x^8 + x^6 + x^5 = x * (x^4 + x^3 + x + 1) + (x^4 + x^3 + x + 1) + x^6 + x^5 = (x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5$

$X^{13} = x^6 + x^3 + x^2 + 1$

$X^{11} = x^3 * x^8 = x^3 * (x^4 + x^3 + x + 1) = \underline{x^7 + x^6 + x^4 + x^3}$

$X^9 = x * x^8 = x * (x^4 + x^3 + x + 1) = \underline{x^5 + x^4 + x^2 + x}$

Por tanto, el resultado de multiplicar los polinomios, se obtiene sustituyendo:

$A \bullet B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$A \bullet B \bmod 2 = (x^6 + x^3 + x^2 + 1) + (x^7 + x^6 + x^4 + x^3) + (x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5 + x^4 + x^3 + 1 \bmod 2$

$A \bullet B \bmod 2 = x^7 + x^6 + 1 = \mathbf{11}00\ 000\mathbf{1} = C\mathbf{1}_{16}$

Como era lógico esperar, el polinomio resultante tendrá grado menor que 8. La multiplicación de polinomios es asociativa y su elemento neutro es el "01". Para cualquier polinomio binario b(x) de grado menor que 8, se puede aplicar el algoritmo extendido de Euclides para calcular un polinomio inverso de b(x). En este caso se habla de "inversa multiplicativa". a(x) es un polinomio inverso de b(x) si:

$$a(x) \bullet b(x) \bmod m(x) = 1 \text{ ó } b^{-1}(x) = a(x) \bmod m(x)$$

es decir, a(x) es la inversa multiplicativa de b(x). El polinomio extendido de Euclides se puede ver como:

$$b(x)a(x) + m(x)c(x) = 1$$

Multiplicación por x.

Se va a analizar un caso interesante, que es la multiplicación de un polinomio por x. Si multiplicamos un polinomio b(x) por x tenemos:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

$$b(x) \bullet x = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Una vez tenemos este resultado se debe realizar la reducción modulo m(x). Si b_7=O el resultado es el mismo polinomio. Si b_7=1, m(x) debe anular el valor de x^8.

En general, para utilizar este tipo de multiplicación los autores definen una función denominada "xtime" que simplifica la multiplicación de un polinomio por potencias de x, este hecho es gracias a que la función xtime se puede ejecutar de forma reiterativa. La función "xtime" consiste en aplicar un desplazamiento a la izquierda al valor que representa el polinomio y una operación or-exclusiva con el valor 0x11B (0x11B=000100011011 = $m(x)=x^8+x^4+x^3+x+1$) cuando el resultado de la multiplicación debe ser reducido módulo m(x). Esta función se puede programar fácilmente de la siguiente manera:

```
int xtime (int valor)
{
        valor = valor << 1;
        if(valor & 0x100)
                valor^=0x11B;
        return valor;
}
```

Ejecutar una vez la función xtime equivale a multiplicar el polinomio representado por su "valor" por x, es decir el polinomio · '02'.

La importancia de esta función recae en su uso reiterado para calcular multiplicaciones de polinomios. Por ejemplo la multiplicación de los siguientes polinomios:

'57' · '13' $(x^6+x^4+x^2+x+1) \cdot (x^4+x+1)$

se puede ver como la multiplicación de '57' por diversas potencias de x, aplicando la propiedad asociativa. Según esto, el polinomio '13' se puede descomponer en potencias de x de la siguiente forma:

'13' = '01' $\oplus$ '02' $\oplus$ '10' $(x^4+x+1) = 1 \oplus x \oplus x^4$

Por lo tanto: '57' · '13' '57' · ('01' $\oplus$ '02' $\oplus$ '10')

'57' $\oplus$ ('57'·'02') $\oplus$ ('57'·'10')

Resolviendo:

'57' • '02' $(x^6+x^4+x^2+x+1) \cdot x$ = xtime(57) = 'AE'

'57' • '04' $(x^6+x^4+x^2+x+1) \cdot x^2$ = (AE) '47'

'57' • '08' $(x^6+x^4+x^2+x+1) \cdot x^3$ = (47) '8E'

'57' • '10' $(x^6+x^4+x^2+x+1) \cdot x^4$ = (8E) '07'

El cálculo de las cuatro llamadas a la función xtime se observa en la siguiente tabla:

Potencia de x	Valor Hexadecimal	Valor Inicial pasado a xtime	Valor desplazado a la izquierda 1 posición	XOR con "Ox11B"	Resultado
x	"02"=00000010b	"57"	"AE"		"AE"
x^2	"04"=00000100b	"AE"	"5C"	5C ⊕ Ox11B=47	"47"
x^3	"08"=00001000b	"47"	"8E"		"8E"
x^4	"10"=00001010b	"8E"	"1C"	1C ⊕ Ox11B=O7	"O7"

Operaciones mediante la función xtime.

Finalmente se calcula:

'57' • '13'= '57' • ('01' ⊕ '02' ⊕ '10') = '57' ⊕ ('57' • '02') ⊕ ('57' • '10') = '57' ⊕ 'AE' ⊕ '07'='FE'

Este procedimiento es interesante ya que las multiplicaciones que se realizan en el algoritmo se pueden realizar utilizando esta función. Su uso se centra en la función MixColumn del algoritmo con polinomios representados por '01','02','03', '09', '0b', '0d' y '0e'.

Consideraciones:

- Multiplicar un polinomio por '01' es igual al mismo polinomio.
- Multiplicar un polinomio por '02' consiste en aplicar la función xtime al polinomio.
- Multiplicar un polinomio A por '03' es igual a:

  (A • '02') ⊕ A = xtime (A) ⊕ A.
- Multiplicar un polinomio A por '09' es igual a:

  (A • '08') ⊕ A = xtime (xtime (xtime (A))) ⊕ A
- Multiplicar un polinomio A por '0b' es igual a:

  (A • '08')⊕ A = xtime (xtime (xtime (A))) ⊕ A
- Multiplicar un polinomio A por '0e' es igual a:

(A•‘02’)⊕ (A• ‘04’) ⊕ (A• ‘08’)= xtime(A) ⊕ xtime(xtime(A)) ⊕ xtime(xtime(xtime (A)))

- Multiplicar un polinomio A por ‘0d’ es igual a:

(A • ‘04’) ⊕ (A • ‘08’) ⊕ A = =xtime (xtime(A)) ⊕ xtime(xtime(xtime (A))) ⊕ A

Especificación del Algoritmo.

El algoritmo Rijndael es un sistema simétrico de cifrado por bloques, por tanto utiliza la misma clave para el proceso de cifrado como para el proceso de descifrado. Su diseño permite la utilización de claves de sistema con longitud variable siempre que sea múltiplo de 4 bytes. La longitud de las claves utilizadas por defecto son 128 (AES-128), 192 (AES-192) y 256 (AES-256) bits. De la misma manera el algoritmo permite la utilización de bloques de información con un tamaño variable siempre que sea múltiplo de 4 bytes, siendo el tamaño mínimo recomendado de 128 bits (el tamaño mínimo de 16 bytes).

Este algoritmo opera a nivel de byte, interpretando éstos como elementos de un cuerpo de Galois GF(2^8), y a nivel de registros de 32 bits, considerándolos como polinomios de grado menor que 4 con coeficientes que son a su vez polinomios en GF(2^8).

Estructura del Algoritmo.

La estructura del algoritmo Rijndael está formada por un conjunto de “rondas”, entendiendo por “rondas” un conjunto de reiteraciones de 4 funciones matemáticas diferentes e invertibles.

Por tanto, el algoritmo se basa en aplicar un número de rondas determinado a una información en claro para producir una información cifrada. La información generada por cada función es un resultado intermedio, que se conoce como Estado, o si se prefiere Estado Intermedio.

El algoritmo representa el “Estado” como un matriz rectangular de bytes, que posee 4 filas y Nb columnas. Siendo el número de columnas Nb en función del tamaño del bloque:

Nb = tamaño del bloque utilizado en bits / 32

Por ejemplo la representación de una matriz de Estado para un tamaño de bloque de 160 bits (Nb = 5), sería:

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$

La clave del sistema se representa con una estructura análoga a la del "Estado", es decir, se representa mediante una matriz rectangular de bytes de 4 filas y N_k columnas.
Siendo el número de columnas N_k en función del tamaño de la clave:

N_k = tamaño de la clave en bits / 32

Por ejemplo la representación de una clave de 128 bits (N_k = 4), en forma de matriz rectangular sería:

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Una vez establecido estos parámetros iniciales el bloque que se pretende cifrar o descifrar se traslada byte a byte sobre la matriz de Estado, siguiendo la secuencia $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, ..., $a_{3,4}$, y análogamente los bytes de la clave se copian en la matriz de la clave siguiendo el mismo criterio, $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$ ... $k_{3,3}$

A partir de este momento la matriz de Estado sufre 4 transformaciones por "ronda" (vuelta), utilizándose en el proceso subclaves para cada ronda que se generan de la clave de sistema elegida.

Las 4 transformaciones que aplica el algoritmo a la matriz de Estado por ronda son:

• Función ByteSub: Sustitución con propiedades óptimas de no linealidad.
• Función ShiftRow y MixColumn: Permiten un alto nivel de difusión de la información a lo largo de las diferentes rondas.
• Función AddRoundKey: Permite aplicar a la matriz de Estado una operación "or exclusiva" con la subclave correspondiente a cada ronda.

El número de reiteraciones o vueltas de las 4 transformaciones sobre la información, o mejor dicho sobre la matriz de Estado Intermedio depende de la versión del algoritmo que se utilice.
Los autores definen que para tamaños de bloques y claves entre 128 y 256 bits (con incrementos de 32 bits) el número de vueltas Nr es determinado por la siguiente expresión:

$$N_r = \max(N_k, N_b) + 6$$

Por ejemplo, para un algoritmo Rijndael de tamaño de clave y de bloque 128 bits, el número de vueltas es 10. Se observa claramente que el número de vueltas o reiteraciones del algoritmo dependen del tamaño de bloque y clave elegidos.

Número de rondas para Rijndael en función de tamaños de clave y bloque:

Clave / Bloque	N_b = 4 (128 bits)	N_b = 6 (192 bits)	N_b = 8 (256 bits)
N_k = 4 (128 bits)	10	12	14
N_k = 6 (192 bits)	12	12	14
N_k = 8 (256 bits)	14	14	14

Teniendo en cuenta la estructura general del algoritmo se va a profundizar, a continuación, en el proceso de cifrado y descifrado.

La mayoría de los cifradores simétricos tienen una estructura tipo Feistel. En esta estructura la vuelta o ronda de transformación consiste en separar un bloque de un mensaje en dos partes, una parte izquierda y otra derecha, e ir conmutando las partes de izquierda a derecha aplicándole una función unidireccional. Este procedimiento reiterado un número de veces constituye la estructura del algoritmo. Muestra de esta estructura se encuentra en el antiguo estándar de cifrado DES.

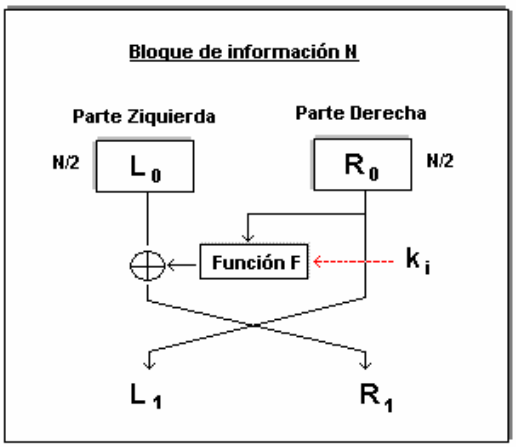

Figura 1: Estructura del Antiguo Estándar de Cifrado DES [11]

Un sistema Feistel procesa la mitad de los bits del bloque en una vuelta quedando el resto inalterados. Rijndael no tiene una estructura tipo Feistel (trata todos los bits por vuelta), en vez de ello se decidió definir la vuelta de transformación como tres funciones invertibles llamadas capas. La elección de las diferentes capas está basada en gran parte en la obra "Cipher and hash function design strategies based on linear and differential cryptanalysis", que permitió a los autores pensar en una estructura de algoritmo que llevara implícita la resistencia contra el criptoanálisis lineal y diferencial. Teniendo en cuenta esto, cada capa cobraría su propia utilidad (estas capas está constituidas por las 4 transformaciones matemáticas del algoritmo):

• Una capa de mezcla lineal, que garantiza una alta difusión de la información a través de la aplicación de varias vueltas.
• Una capa no lineal, que permita propiedades aptas de no linealidad.
• Una capa de adición de clave, que consiste en una simple operación "or exclusiva" entre la subclave de cada vuelta y el Estado intermedio.

Se puede observar como el algoritmo utiliza aparte de las rondas estándar una ronda final y otra inicial. La ronda inicial es útil para aplicar una subclave antes de la primera vuelta. El motivo de aplicar esta clave consiste en que no se pueden atacar directamente a las capas sin conocimiento de la clave. Tal es esta propiedad que se utiliza en otros algoritmos como el algoritmo IDEA, SAFER y Blowfish. Por otro lado para que el algoritmo cifrador y el descifrador tengan una estructura lo más similar posible, la

capa de mezcla lineal de la última vuelta es diferente de la capa de mezcla lineal de las otras vueltas, este es el motivo por el cual la ronda final difiere de la ronda estándar.

Descripción del proceso de cifrado.

El proceso de cifrado consiste en la aplicación de 4 funciones matemáticas invertibles sobre la información que se desea cifrar. Estas transformaciones se realizan de forma reiterativa para cada ronda o vuelta definida.

Gráficamente la descripción del proceso de cifrado con el algoritmo Rijndael se puede ver como:

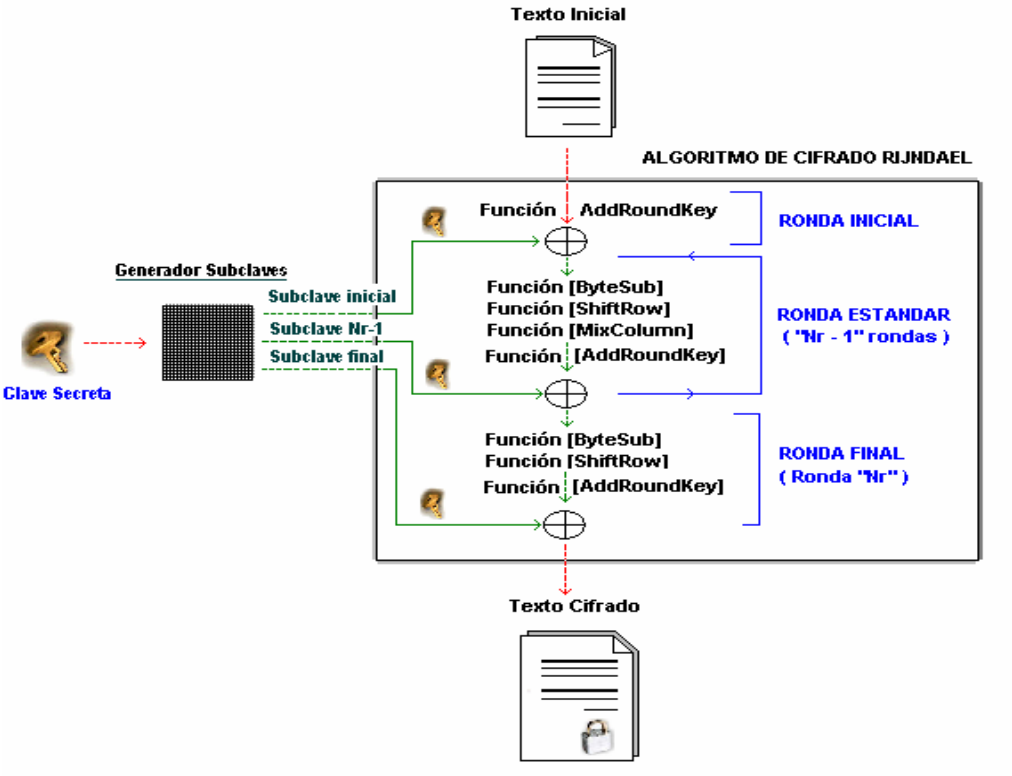

Figura 2: Descripción del Proceso de Cifrado [11]

En esencia la información a cifrar se va mapeando en la matriz de Estado. Esta matriz de Estado se introduce al cifrador, y sufre una primera transformación, en la ronda inicial, que consiste en una operación or- exclusiva (AddRoundKey) entre una Subclave generada y la matriz de Estado. A continuación, a la matriz de Estado resultante se le aplican 4 transformaciones invertibles, repitiéndose

este proceso "N_r-1" veces, en lo que se conoce como Ronda Estándar. Finalmente se le aplica una última ronda o vuelta a la matriz de Estado resultante de las "N_r-1" rondas anteriores, aplicando las funciones ByteSub, ShiftRow y AddRoundKey en este orden. El resultado de la ronda final produce el bloque cifrado deseado.

En esta figura se puede observar la evolución del cifrador, y como entra en juego la clave del usuario generando subclaves que se utilizan para cada ronda.

A continuación vamos a profundizar en cada una de las funciones que configuran el algoritmo cifrador Rijndael para comprender mejor su funcionamiento.

Motivos de diseño

En este apartado se van a discutir los parámetros de diseño del algoritmo Rijndael, analizando sus motivos de elección y la influencia que tienen sobre la seguridad del algoritmo. La base de los siguientes razonamientos, se encuentra en un estudio que sus autores publicaron al respecto. Teniendo en cuenta esto el algoritmo Rijndael se estructura en torno a 3 axiomas:

- Resistencia contra todos los ataques conocidos. **[11]**
- Rapidez y compatibilidad en un gran número de plataformas.
- Sencillez en el diseño.

Teniendo en cuenta estos principios se va a profundizar en los motivos de diseño de las funciones matemáticas que utiliza el algoritmo, el número de vueltas y su estructura algebraica. **[11]**

Seguridad del algoritmo

Hoy en día muchas personas manejan el concepto de seguridad de una manera muy relativa, el lenguaje usado es particular y no se tiene en general conceptos bien definidos. Sin embargo en los últimos 30 años se han podido resumir varios puntos básicos para que un algoritmo simétrico sea "seguro".

Hay quienes dicen que aún el diseño de un algoritmo criptográfico es más ingeniería que ciencia. En términos generales un algoritmo es seguro si éste está diseñado para soportar todos los ataques conocidos hasta el momento y da la suficiente evidencia de su fortaleza. Aunque es claro que siempre

existe la posibilidad de que el algoritmo pueda ser roto por recientes y novedosos ataques, es decir, es imposible diseñar un algoritmo inmune a ataques no conocidos.

Desde el punto de vista metodológico el algoritmo debió de haberse sometido en un tiempo considerable al análisis y estudio de la comunidad científica.

En el caso concreto de un algoritmo criptográfico simétrico, existen varios comportamientos que nos dirán si un algoritmo puede considerarse seguro. El algoritmo debe de dar evidencia de haber pasado la mayoría de los análisis que pudieran existir. En términos más técnicos un algoritmo simétrico debe de dar evidencia de ser inmune a los ataques más potentes y conocidos, en este caso al menos al análisis lineal y el análisis diferencial.

AES posee características para poder evitar ataques como el lineal y diferencial.

Las características más mencionadas que debe de contar un algoritmo simétrico son:

1. La no linealidad entre las entradas y las salidas, o la correlación de las entradas con las salidas.
2. La propagación de las diferencias de los bits, o el medir la probabilidad de que tanto se confunden los bits.

Particularmente las anteriores características son las más requeridas en un algoritmo simétrico, en el diseño las dos se mezclan y se miden en cada una de las rondas que consiste el algoritmo, es decir, en términos muy generales y básicos si un algoritmo tiene esas dos propiedades y se realizan las suficientes rondas, entonces el algoritmo será inmune a los análisis lineal y diferencial.

Enseguida trataremos de explicar de la manera más sencilla posible las anteriores dos características.

1. LINEALIDAD

En términos elementales por ejemplo si las entradas de nuestro algoritmo son 1, 2, 3, 4, y 5 y tenemos como salidas 2, 4, 6, 8, y 10, claramente hay una dependencia lineal entre las entradas y las salidas, es decir $f(x)=2g(x)$ donde f,g son las funciones de entrada y salida correspondientemente. El método conocido como correlación es el que se aplica a dos conjuntos de datos A, B y determina que tanto pueden haber uno del otro una dependencia lineal. De tal modo que si existe una relación lineal entre las

entradas de las salidas, no es nada difícil conocer la información de los textos originales o de la clave de cifrado. Claramente la linealidad no es una propiedad que se quiera en un algoritmo criptográfico.

El mecanismo que impide que haya correlación es el alternar las claves, es decir que para cada ronda de nuestro algoritmo se aplique una clave diferente, esto se logra en términos generales implementado un programa de claves que proporciona una clave diferente para cada ronda. Esto permite disminuir la linealidad en la mayoría de las modalidades que pueda pensarse como una debilidad de nuestro algoritmo, y es aprovechada principalmente por el criptoanálisis lineal.

2. PROPAGACIÓN

El otro concepto muy trabajado en una algoritmo simétrico es la propagación, este concepto hay que trabajarlo de la manera más cuidadosa, el que el algoritmo tenga buena "propagación" impide que sea aplicado principalmente el criptoanálisis diferencial que es un ataque del tipo "Chosen Plaintext Attack" y permite derivar información de la clave a partir de conocer las probabilidades de las diferencias de la propagación, por lo que estas probabilidades deben de ser lo más pequeño posible. La propagación se obtiene con el programa de claves, con la propagación de la función no lineal del algoritmo (S-Box), el número de rondas, etc.

Cada uno de los elementos de AES fue cuidadosamente elegido para poder cumplir al menos con los dos requerimientos básicos anteriores. Por otra parte otro tipo de ataques o conceptos de debilidad que han sido propuestos, Rijndael los evita, como "Square Attack", "Six Round Attack", "Herds Attack", "Gilbert-Minier Attack", "Interpolation attack", "Related-Key Attack", "Timing Attack", "Imposible Differential attack", "Collision attack", últimamente se han propuesto los llamados ataques algebraicos que en general explotan las propiedades algebraicas del algoritmo. Sin embargo por el momento no se ha podido montar un ataque a la versión completa de Rijndael, lo que lo hace tan seguro como una búsqueda exhaustiva.

Últimos ataques a AES

Describamos ligeramente algunos de los últimos ataques, que de alguna manera son versiones modificadas del los ataques diferencial y lineal.

Impossible Differentials Attack: existe un ataque de este tipo a 5 rondas de AES, requiriendo 229 plaintext elegidos, 230 encripciones, 242 bytes de memoria, 226 pasos de precálculo. Estas condiciones fueron mejoradas para alcanzar un ataque a 6 rondas de AES.

Square Attack: el más potente ataque a Rijndael a la fecha es el ataque "Square", es un ataque dirigido a un algoritmo del tipo de Rijndael que basa su diseño en estructuras de bytes. Precisamente el primer ataque de este tipo fue hecho al algoritmo predecesor llamado "Square". Este ataque puede romper a Rijndael de 6 a 7 rondas, que puede ser mejorado para atacar a 9 rondas de AES-259 con 277 plaintexts, con 256 claves relacionadas, y 2224 inscripciones.

Collision Attack: este ataque afecta a todas las versiones de AES, 128, 192 y 256 con 7 rondas.

Los ataques anteriores son de alguna manera el "último" intento de diseñar un ataque a AES de la manera tradicional. Es obvio que la introducción de estructuras algebraicas a AES por un lado deja atrás a los ataques más tradicionales, pero por el otro reta a encontrar nuevos ataques dirigidos a la nueva estructura algebraica.

De manera natural los nuevos ataques son llamados "ataques algebraicos", y como casi siempre en estos temas existen dos tendencias una de ellas que dice que solo son ideas que pueden o no pueden considerarse aún como peligrosas, y la otra que dice que este tipo de ataques promete mucho.

En general este tipo de ataques consiste en dos etapas: la primera de coleccionar datos como los plaintexts, los ciphertexts, las claves, valores intermedios, y expresa todo el algoritmo como ecuaciones que involucran los datos anteriores. La segunda es resolver las ecuaciones, donde la solución deberá ser información de la clave.

Precisamente debido al diseño tan "formal" de AES, éste puede ser expresado de diferentes maneras de una forma elegante.

Algunas "ideas" o "ataques" algebraicas/os pueden ser listados ahora:

Fracciones continuas: una de las primeras propuestas propone una sola fórmula que puede ser vista como una fracción continua como la siguiente:

$$x = K + \sum \cfrac{C_1}{K' + \sum \cfrac{C_2}{K' + \sum \cfrac{C_3}{K' + \sum \cfrac{C_4}{K' + \sum \cfrac{C_5}{K' + \cfrac{}{K' + P:}}}}}}$$

Donde cada K es un byte que depende de varios bytes de la clave expandida, los Ci son constantes conocidas, y los * son exponentes o índices desconocidos, estos valores depende de la suma que se efectúan. Una combinación de este tipo de de fórmulas describe totalmente al algoritmo AES, con 226 incógnitas, sin embargo no se conoce un método práctico que resuelva este tipo de ecuaciones.

XSL: Los autores observaron que las S-cajas de AES pueden ser descritas por ecuaciones cuadráticas booleaneas, es decir, si x1, x2,..., x8 son los bits de entrada y y1, y2,..., y8 los bits de salida entonces existe una función de la forma f(x1, x2,..., x8, y1, y2,..., y8)=0, donde el grado de f es 2.

El ataque se fundamenta que este tipo de ecuaciones sirven para el segundo paso del ataque algebraico y que existe un método que pueda resolverlas. Este problema está considerado del tipo “Multivariate Quadratic Equations” que se sabe es un problema difícil de resolver. Sin embargo algunas opiniones mencionan que el trabajo de Courtois y Pieprzyk tiene imperfecciones, particularmente que no cuentan con las ecuaciones lineales suficientes para resolver el sistema. La complejidad estimada para el ataque en el mejor de los escenarios es de 2255 lo que equivale a la resistencia de la versión AES-256.

Embedding: otra idea que fue expuesta es la de Murphy y Robshaw, tratando de encajar a AES en otra algoritmo llamado BES(Big Encryption System), de la siguiente manera:

$$Rijndael_K(x) = \phi^{-1}(BES\phi_{(K)}(\phi(x)))$$

donde K es la clave, x el mensaje y f un mapeo de la estructura de campo de Rijndael a la estructura de campo de BES. Como Rijndael usa al combinaciones de los campos GF(2) y GF(28), BES sólo usa el campo GF(28). Los autores afirma que BES tiene mejor estructura algebraica y pudiera ser más fácil su criptoanálisis, sin embargo no se puede afirmar que sus propiedades puedan ser trasladadas a Rijndael. Se afirma aquí también que el método XSL puede ser aplicado a BES con ligeras mejorías en los resultados.

Dual Cipher: Otra manera de encajamiento es definir un algoritmo dual a AES, esto quiere decir una especie de traslación. Si tenemos los mapeos invertibles f, g, h entonces el Cipher "Dual" es definido como:

$$Rijndael_K(x) = f^{-1}Dual_{g(K)}(h(x))$$

Por ejemplo, como la función cuadrado es lineal en los campos de característica 2, es natural que los duales inmediatos sean los cuadrados.
Aunque se puede mostrar que los duales son equivalentes en seguridad, la idea es poder encontrar un ataque a algún dual para que después sea trasladado al original Rijndael, sin embargo por el momento no se ha encontrado alguna debilidad llamativa. **[28][29][30]**

1.5.2- Algoritmos Asimétricos.

Los algoritmos asimétricos más conocidos son:

ElGamal: El algoritmo citado puede emplearse también para cifrar información, su complejidad y el hecho de duplicar el texto cifrado la longitud del texto en claro lo hacen poco recomendable para tal fin.
Algoritmo criptográfico de clave pública que puede emplearse tanto para cifrar como para obtener firmas digitales y que basa su seguridad en la dificultad de calcular logaritmos discretos en un campo finito. **[7][8]**

Rabin: Este algoritmo es relativamente rápido para la búsqueda de cadena de caracteres. En promedio, el tiempo de ejecución es lineal con respecto a la longitud de la entrada. Se basa en la utilización de funciones hash para comparar cadenas.

Un modelo simple (e ineficiente) de función de hash es:

f(x) = 0 para todo entero x.

Obviamente, la colisión hash en esta función es total. Una un poco más interesante es:

f(x) = x mod 1021

Esta función devuelve el resto de la división x entre 1021. Obviamente, la colisión es menor siempre que el conjunto del cual toma valores x no sea muy grande o lo suficientemente aleatorio. Además, nótese que el hecho de que 1021 sea un número primo no es algo azaroso sino que fue cuidadosamente elegido ya que mecanismos que utilizan este tipo de funciones con números primos como base son muy comunes en criptografía. **[7][8]**

RSA: El sistema criptográfico con clave pública RSA es un algoritmo asimétrico cifrador de bloques, que utiliza una clave pública, la cual se distribuye (en forma autenticada preferentemente), y otra privada, la cual es guardada en secreto por su propietario.

Una clave es un número de gran tamaño, que una persona puede conceptualizar como un mensaje digital, como un archivo binario o como una cadena de bits o bytes.

Cuando se quiere enviar un mensaje, el emisor busca la clave pública de cifrado del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, éste se ocupa de descifrarlo usando su clave oculta.

Los mensajes enviados usando el algoritmo RSA se representan mediante números y el funcionamiento se basa en el producto de dos números primos grandes (mayores que 10100) elegidos al azar para conformar la clave de descifrado.

Emplea expresiones exponenciales en aritmética modular.
La seguridad de este algoritmo radica en que no hay maneras rápidas conocidas de factorizar un número grande en sus factores primos utilizando computadoras tradicionales.
La computación cuántica podría proveer una solución a este problema de factorización.

Este popular sistema se basa en el problema matemático de la factorización de números grandes**.**

El algoritmo RSA funciona de la siguiente manera:

1. Inicialmente es necesario generar aleatoriamente dos números primos grandes, a los que llamaremos p y q.
2. A continuación calcularemos n como producto de p y q: n = p * q
3. Se calcula fi: fi(n)=(p-1)(q-1).
4. Se calcula un número natural e de manera que MCD (e, fi(n))=1, es decir e debe ser primo relativo de fi(n).
5. Es lo mismo que buscar un número impar por el que dividir fi(n) que de cero como resto.
6. Mediante el algoritmo extendido de Euclides se calcula d: ed. mod fi(n)=1 Puede calcularse d=((Y*fi(n))+1)/e para Y=1,2,3,... hasta encontrar un d entero.
7. El par de números (e,n) son la clave pública.
8. El par de números (d,n) son la clave privada.
9. Cifrado: La función de cifrado es C = M^e mod n
10. Descifrado: La función de descifrado es M = C^d mod .**[7][8]**

1.6- Características Generales de los algoritmos criptográficos

Las principales características que un sistema de seguridad quiere obtener son:

• Confidencialidad. Consiste en garantizar que sólo las personas autorizadas tienen acceso a la información.
• Integridad. Consiste en garantizar que el documento original no ha sido modificado. El documento puede ser tanto público como confidencial.
• Autenticación. Permite garantizar la identidad del autor de la información.

Existen diversos algoritmos matemáticos que intentan cubrir una o varias de estas características básicas de seguridad. El nivel de cumplimiento de sus objetivos es difícil de evaluar, ya que diversos algoritmos pueden ser vulnerables ante técnicas de ataque diferentes, además la mayoría de los algoritmos pueden trabajar con claves de distinta longitud lo cual afecta directamente a la robustez. Por otro lado, existen otras características, aparte de la robustez del algoritmo, que también influyen en el proceso de selección del algoritmo más apropiado para una determinada aplicación. Algunas de estas características son: el tiempo de cálculo del proceso de cifrado, el tiempo de cálculo del proceso de descifrado, la relación de tamaño entre el documento original y el documento cifrado. **[9]**

CAPÍTULO #2. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

Luego de lo visto en el capítulo anterior, se cuenta con el conocimiento acerca de una serie de algoritmos de encriptación y desencriptación para asegurar los recursos multimedia de aplicaciones gráficas en Entornos de Realidad Virtual como son: IDEA (International Data Encryption Algorithm), DES(Data Encryption Standard), Triple-DES(T-DES),BlowFish, Rijndael (El nuevo estándar AES) por solo citar algunos. Con lo que queda demostrado que la implementación de alguno de estos métodos resulta bastante costosa en su procesamiento, sirviendo este conocimiento como premisa en la solución que se propone a continuación.

2.1- Soluciones Técnicas

Después del análisis de los diferentes algoritmos de encriptación se ha planteado el uso del algoritmo Advanced Encryption Standard (AES), también conocido como Rijndael ya que la seguridad del mismo fue el aspecto fundamental que se tuvo en cuenta para la selección, pues hoy en día, es uno de los algoritmos de mayor seguridad y confianza por su grado de fortaleza, velocidad y bajo consumo de recursos.

Algoritmo Advanced Encryption Standard (AES), también conocido como Rijndael.

Se pudo constatar que algoritmos como el MARS, RC6, SERPENT, TWOFISH, IDEA, TRIPLE-IDEA entre otros poseen un margen de seguridad adecuado sin embargo han recibido críticas debido a su bajo margen de seguridad, y debido a que muchos tienen un grado de complejidad bastante alta. Sin embargo se demostró que el algoritmo RIJNDAEL tiene un margen de seguridad alto el cual es bastante difícil de medir debido a que el número de rondas cambia con el tamaño de la clave, su estructura es bastante simple lo que ha facilitado su análisis de seguridad durante el tiempo especificado en el proceso de desarrollo del AES.

Otro tema muy importante fue el estudio de la velocidad con que se ejecutaba cada algoritmo, pues la misma puede medirse entonces por cuantos millones de bits por segundo procesa. Los tres procesos más comunes en un algoritmo son el cifrado, el descifrado y el programa de claves. La realización de MARS, RC6 y SERPENT no varían significativamente para los tres tamaños de claves de AES. Para Rijndael y Twofish, sin embargo la configuración de la clave es lógicamente, más lento para claves de 192 bits que para claves de 128 bits, y más lento todavía para claves de 256 bits, aunque en estos casos ofrecen una compensación en el incremento de la seguridad.

La eficiencia que presenta este algoritmo, frente al resto de los algoritmos consume menos memoria de los equipos y es muchísimo más rápido. La eficiencia además de depender del diseño del algoritmo está directamente relacionada con el hardware.

La eficiencia en software de RIJNDAEL radica en sus funciones básicas. La operación más costosa en tiempo, es el obtener inversos multiplicativos del campo GF(2^8), en este caso, esta operación es precalculada y la operación es substituida por una consulta de S-Box, con esta misma técnica varios

cálculos del algoritmo son precalculados y entonces se evitan los cálculos reemplazándolos por consultas de tablas.

A continuación se muestra una tabla comparativa entre algunos algoritmos basándonos en los siguientes factores de comparación: Seguridad, Eficiencia Computacional, Requisitos de Memoria, Simplicidad de Diseño y Flexibilidad.

	Seguridad	**Eficiencia Computacional**	**Simplicidad de Diseño**	**Flexibilidad**
MARS	Margen de seguridad adecuado	Rápido	Complejo	No
RC6	Margen de seguridad adecuado	Rápido	Simple	Si
RIJNDAEL	Margen de seguridad alto	Rápido	Simple	Si
SERPENT	Margen de seguridad adecuado	Rápido	Simple	Si
TWOFISH	Margen de seguridad adecuado	Lento	Complejo	No

Tabla 1: Comparación de Algoritmos Criptográficos

Rijndael se presentó como el algoritmo más rápido en multitud de plataformas: 32 bits, procesadores de 8 bits, etc. **[11]**

Se realizó un estudio en diferentes plataformas con diferentes características, de acuerdo a las distintas propiedades que cada una posee, para determinar y realizar una comparación de la velocidad de procesamiento en el proceso de cifrado.

Plataformas:

- Intel Core 2 Quad CPU 2.40 GHz.

- Intel Pentium CPU 3.00 GHz.

- Intel Pentium Dual CPU 2.00 GHz.

- Intel Pentium 4 CPU 3.00 GHz.

- Celeron 1.80 GHz.

No.	Fabricante	Tipo	Velocidad Procesador	Resultado en milisegundos
1.-	INTEL	Pentium Dual CPU	2.00 GHz	2563
2.-	INTEL	Core 2 Quad CPU	2.40 GHz	875
3.-	INTEL	Pentium 5 CPU	3.00 GHz	2921
4.-	INTEL	Pentium 4 CPU	3.00 GHz	9059
5.-		CELERON	1.80 GHz	22386

Tabla 2: Pruebas de Velocidad de Cifrado

Los resultados obtenidos, están basados sobre pruebas que se realizaron a los 94 archivos de texturas predeterminados, que presenta el motor de render OGRE, sobre el algoritmo Rijndael implementado en C++; y como resultado se obtiene un gran desempeño del algoritmo, mostrando una gran velocidad de procesamiento de cifrado, dada las características de la plataforma en que se hace uso.

2.2- Herramientas de desarrollo y lenguaje utilizado.

Las herramientas que se proponen en este epígrafe serán puestas en marcha en todas las fases de desarrollo del módulo, tal como el lenguaje que se referencia.

2.2.1- Editor de Código Visual Studio 2008

Se propone la utilización del IDE Visual Studio 2008, solamente como editor de código, dado su fácil manejo e interfaz, y a su módulo integrado de ayuda y completamiento de código: Visual Assist. Visual Studio 2008 facilita que los diseñadores y los creadores de código trabajen de forma paralela y conjunta

sobre un proyecto. La productividad sin duda es uno de las ventajas indiscutibles de .NET. Funcionalidades que en el pasado requerían miles de líneas de código ahora pueden implementarse de forma muy sencilla y rápida usando las piezas que el propio framework suministra. El IDE ofrece una productividad sin precedentes, dando soporte no solo a las tareas habituales de un desarrollador sino a todas las relacionadas con calidad del software y gestión del ciclo de vida del desarrollo.Se harán uso de bibliotecas estándares para enfocarlo a un esquema multiplataforma. **[24]**

2.2.2- Rational Rose

El Rational Rose es una herramienta CASE profesional que emplea el UML como lenguaje de modelado y además es la notación estándar para arquitectura de software y soporta el ciclo completo de vida del desarrollo de un software. Esto significa que con Rational Rose, todo el equipo puede comunicarse con un lenguaje y una herramienta. Rational Rose domina el mercado de herramientas para el análisis, modelamiento, diseño y construcción orientado a objetos. De acuerdo a International Data Corporation (IDC), la participación de Rational en 1998 con respecto a los ingresos del mercado es mayor que las participaciones de los siguientes cuatro competidores directos combinados. Por cuatro años consecutivos IDC ha nombrado a Rational Rose como "La Herramienta Líder en Análisis, Diseño y Construcción Orientada a Objetos", con base en los ingresos del producto. Rational Rose permite visualizar, entender, y refinar los requerimientos y arquitectura antes de enfrentar el código. Esto permite, evitar esfuerzos desperdiciados en el ciclo de desarrollo. El modelo arquitectónico puede ser rastreado hacia el modelo de procesos de negocios y los requerimientos de sistema.

Esta herramienta permite especificar, analizar, diseñar el sistema antes de codificarlo. Tiene varias características que lo distinguen como son el de chequear la sintaxis UML, mantiene la consistencia de los modelos del sistema del software, genera la documentación automáticamente, la generación de código a partir de los modelos, permite la ingeniería inversa (crear modelo a partir código) entre otras. **[24]**

2.2.3- C++

Se codificará en C++, dado que es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia, además es multiplataforma.

2.2.4- Motor Gráfico OGRE

OGRE (Object Oriented Graphics Engine) es un motor de gráficos en tres dimensiones multiplataforma. Al tener esta característica posee importantes ventajas y es que no existen problemas para quienes quieran crear aplicaciones para Windows, Linux o Mac. Esto es posible gracias a que está construido de forma que no se compromete con una API en particular, puesto que el motor soporta tanto el uso de DX9 como de OpenGL. Además, la principal ventaja de OGRE sobre otros engines 3D es que es un proyecto open source bajo licencia LGPL. Esto significa que su uso es gratuito y apenas existen exigencias para su uso y específicamente representa la herramienta principal de nuestro proyecto. Se puede utilizar OGRE también para hacer aplicaciones de simulación, corporativas o de investigación. **[25][26]**

2.2.5- Qt 4.6

Qt son un conjunto de bibliotecas multi-plataforma para el desarrollo del esqueleto de aplicaciones GUI, escritas en código C++. Qt además está completamente orientado a objetos. A continuación se exponen las características que presenta:

- QT es una biblioteca para la creación de interfaces gráficos. Se distribuye bajo una licencia libre GPL (o QPL) que nos permite incorporar las QT en nuestras aplicaciones open-source
- Se encuentra disponible para una gran número de plataformas: Linux, MacOs X, Solaris, HP-UX, UNIX con X11. Además, existe también una versión para sistemas empotrados
- Es orientado a objetos, lo que facilita el desarrollo de software. El lenguaje para el que se encuentra disponible es C++ aunque han aparecido bindings a otros lenguajes como Python o Perl
- Es una biblioteca que se basa en los conceptos de widgets (objetos), Señales-Slots y Eventos (ej: clic del ratón).
- Las señales y los slots es el mecanismo para que unos widgets se comuniquen con otros.
- Los widgets pueden contener cualquier número de hijos. El widget "top-level" puede ser cualquiera, sea ventana, botón, etc.
- Algunos atributos como el texto de etiquetas, etc. ... se modifican de modo similar al lenguaje html
- QT proporciona además otras funcionalidades:
    - Biblioteca básica -> Entrada/Salida, Manejo de Red, XML
    - Interface con bases de datos -> Oracle, MySQL, PostgreSQL, ODBC

- Plugins, biblioteca dinámicas (Imágenes, formatos, ...)
- Unicode, Internacionalización **[35]**

2.2.5.1- Qt Designer

Es una herramienta muy potente que permite diseñar de una forma muy sencilla y rápida ventanas de dialogo con las biblioteca Qt. Esta herramienta es una aplicación mediante la cual se puede realizar el diseño de aplicaciones GUI de forma gráfica y muy intuitiva. **[35]**

2.3- Modelo de Domino

Se propone el modelo de dominio correspondiente a la Figura 3. En el que se representan los conceptos fundamentales asociados al trabajo con los algoritmos de encriptación. Además se presentan los eventos que suceden en el entorno en el que trabaja el sistema, y se muestran también los objetos más importantes en el contexto del sistema.

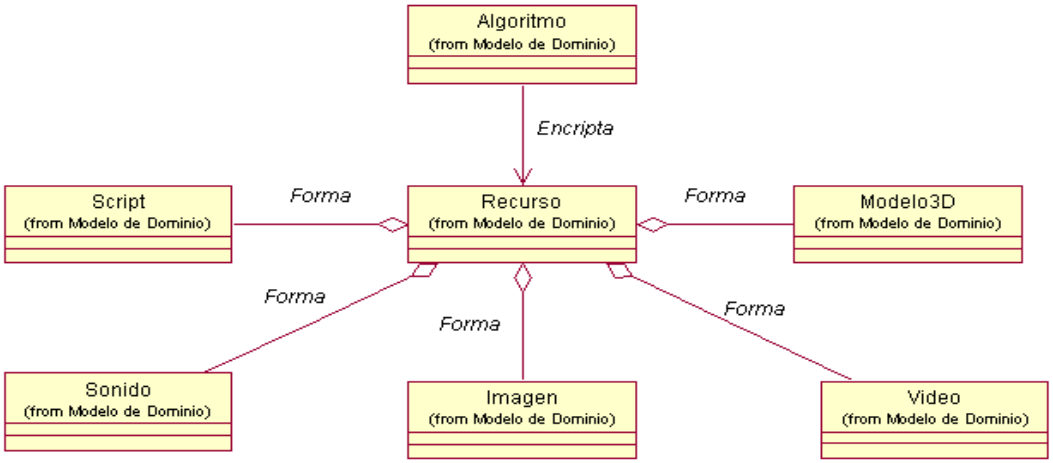

Figura 3: Modelo de Dominio

2.4- Glosario de Términos del Modelo de Dominio

Algoritmo: Conjunto de acciones o secuencias de operaciones ejecutadas en un determinado orden para resolver un problema.
Recurso: En informática se llaman recursos a los medios utilizados por los dispositivos para ejecutar sus funciones, provistos por los elementos del ordenador.
Modelo 3D: Modelo tridimensional, real o virtual para representar el cuerpo o una parte del mismo, como en el caso de los ARPATIs (Pacientes artificiales).
Imagen: Archivo codificado, que al abrirlo, muestra una representación visual (ya sea fotografía, gráfica, dibujo, entre otras).
Script: Un script es un guión o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades. Es muy utilizado para la administración de sistemas UNIX. Son ejecutados por un intérprete de línea de comandos. Usualmente son archivos de texto.
Sonido: Desde un punto de vista físico, el sonido es una vibración que se propaga en un medio elástico (sólido, líquido o gaseoso), cuando nos referimos al sonido audible por el oído humano, lo definimos como una sensación percibida en el órgano del oído, producida por la vibración que se propaga en un medio elástico en forma de ondas.
Video: Fichero con una extensión determinada que contiene datos de audio e imagen.

2.5- Reglas del Negocio

Las reglas del negocio no son más que parámetros o restricciones por los que se rige el sistema, siendo necesaria las mismas para el desarrollo de la solución informática.

1.- El algoritmo puede encriptar cualquier tipo de recurso, pero solo se va a utilizar para el trabajo con modelos 3D, scripts y texturas.
2.- Los recursos deben ser encriptados con anterioridad.
3.- Los recursos serán desencriptados por la aplicación final.

2.6-Requeriemientos del sistema

A continuación se hace referencia a los requisitos funcionales y no funcionales más significativos capturados para el desarrollo del sistema.

2.6.1- Requisitos funcionales.

2.6.1.1- Requisitos funcionales para encriptar.

1.- Abrir archivo.
2.- Mostrar mensaje de error si no se puede abrir el recurso.
3.- Leer en memoria los datos del recurso.
4.- Mostrar mensaje de error si los datos del recurso no pueden ser leídos.
5.- Definir las dimensiones del archivo.
6.- Crear buffer.
7.- Guardar características del archivo en el buffer.
8.- Definir tamaño de la llave de cifrado.
9.- Crear tablas de cifrado.
10.- Verificar integración de las tablas.
11.- Inicializar tablas.
12.- Volcar el contenido de la tabla.
13.- Expandir llave de cifrado.
14.- Definir bloques de cifrado.
15.- Definir modo de bloque.
16.- Separar bloques de mensajes.
17.- Conmutar bloques.
18.- Aplicar función unidireccional.
19.- Encriptar bloque.
20.- Pasar al siguiente bloque.
21.- Llenar el buffer con los bloques encriptados.
22.- Integrar bloques encriptados.
23.- Crear fichero encriptado.
24.- Eliminar buffer.
25.- Cerrar archivo.

2.6.1.2- Requisitos funcionales para desencriptar.

26.- Abrir recurso encriptado.

27.- Mostrar mensaje de error si no se puede abrir el recurso.
28.- Verificar existencia de llave.
29.- Mostrar mensaje de error si no se encuentra la llave.
30.- Verificar la validez de la llave.
31.- Definir bloques encriptados.
32.- Expandir la llave de cifrado.
33.- Aplicar función de desencriptación de bloques.
34.- Liberar tablas de cifrado.
35.- Generar archivo de salida.
36.- Cerrar archivo.
37.- Integrar dirección de archivo al motor gráfico OGRE.

2.6.2- Requisitos no funcionales

- Usabilidad: Los futuros usuarios del sistema serán programadores con conocimientos básicos con respecto al tema de criptografía.
- Portabilidad: Multiplataforma.
- Rendimiento: Debe ser una aplicación que tenga gran alto de velocidad de procesamiento de los cálculos, tiempo de respuesta y recuperación del sistema.
- Software: Sistema Operativo Linux o Windows.
- Diseño e Implementación: Se regirá por la filosofía de Programación Orientada a Objetos y se codificará con el lenguaje C/C++.

2.7- Diagramas de Casos de Uso

Se propone los siguientes diagramas de casos de uso que representan la Figura 4 y Figura 5, los cuales agrupan todos los requisitos funcionales.

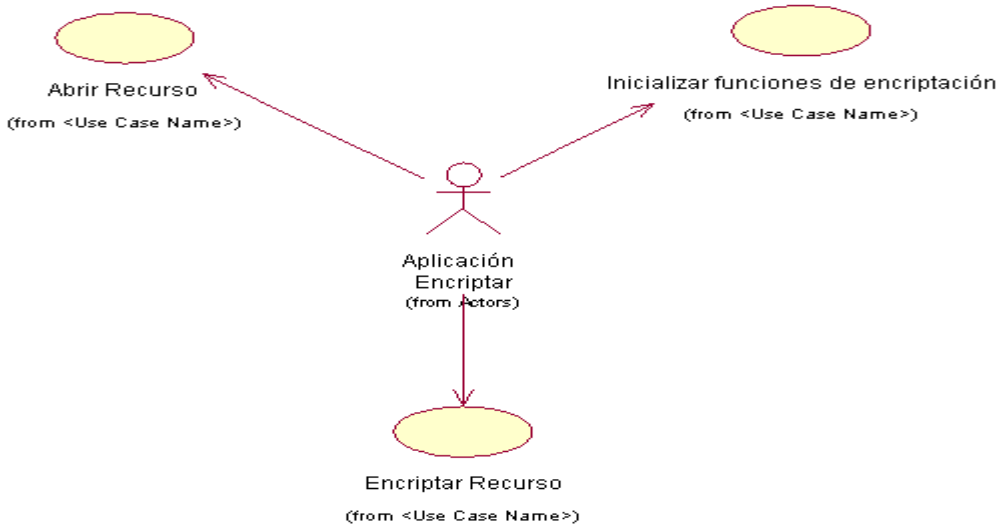

Figura 4: Diagrama de Casos de Uso Aplicación Encriptar

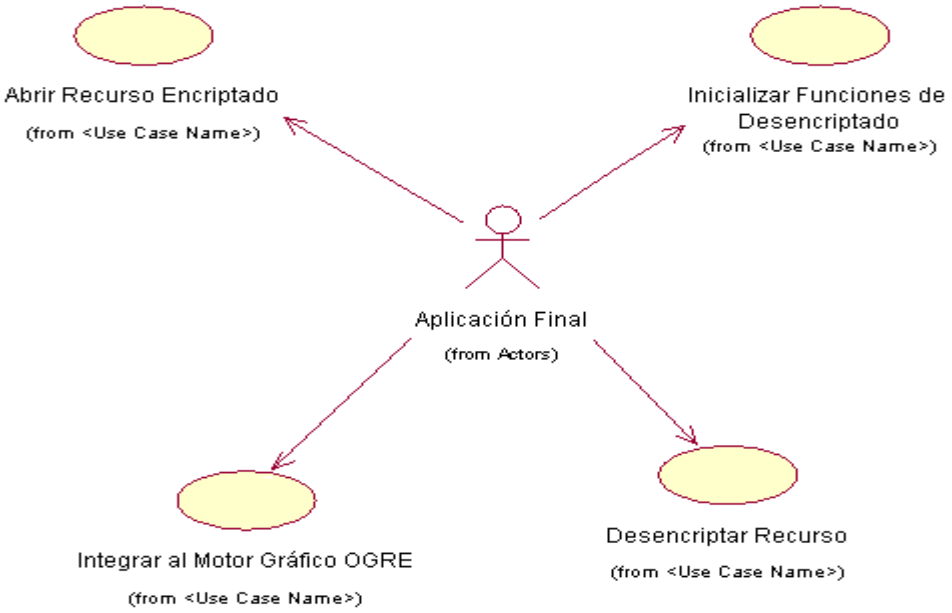

Figura 5: Diagrama de Casos de Uso Aplicación Final

2.8- Expansión de los Casos de Uso

2.8.1- Definición de los actores

Actores	**Justificación**
Aplicación Encriptar	La aplicación que utilice este método de encriptación se beneficiará con las funcionalidades que el mismo brinda.
Aplicación Final	La aplicación gráfica que utilice las funciones de desencriptación, se favorecerá con las ventajas que provee.

Tabla 3: Definición de los actores

2.8.2- Casos de Uso Expandidos

Nombre del Caso de Uso		Abrir Recurso
Actores	Aplicación Encriptar	
Propósito	Abrir los recursos	
Resumen: El caso de uso se inicia cuando la aplicación solicita abrir un recurso, luego de verificar que el mismo existe.		
Precondiciones		Que exista el recurso
Referencias		R1,R2
Curso Normal de Eventos:		
Acción del Actor		Respuesta del Sistema
1- Solicita abrir un recurso especificando la dirección		2- Se verifica que el recurso existe.
Curso Alternativo de los eventos:		
		2.1- Si el recurso no existe se muestra un mensaje de error.
Poscondiciones:		La información del recurso debe quedar almacenada.

Tabla 4: Expansión del Caso de Uso Cargar Recurso

Nombre del Caso de Uso	Inicializar Funciones de Encriptación
Actores	Aplicación Encriptar
Propósito	Inicializar funciones para efectuar el proceso de encriptación.
Resumen: El caso de uso se inicia cuando la aplicación solicita la inicialización de las funciones necesarias para iniciar el proceso de encriptación.	
Precondiciones	Que el recurso haya sido cargado.
Referencias	R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15.
Curso Normal de Eventos:	
Acción del Actor	Respuesta del Sistema
1-Solicita la lectura de los datos del recurso	2- Se verifica que el recurso se ha cargado correctamente.
	3- Se definen las características del recurso.
	4- Se crea el buffer que almacenará los datos del recurso y se llena con los datos del recurso.
	5- Se define la llave para crear las tablas de cifrado. Se verifica su integración y se inicializan las mismas.
Curso Alternativo de los eventos:	
	2.1- Si el recurso no fue cargado se muestra un mensaje de error.
Poscondiciones:	

Tabla 5: Expansión del Caso de Uso Inicializar Funciones de Encriptación

Nombre del Caso de Uso	Encriptar Recurso
Actores	Aplicación Encriptar
Propósito	Encriptar recurso para proteger su información.
Resumen: El caso de uso se inicia cuando la aplicación solicita la encriptación de los bloques de información del recurso a encriptar.	
Precondiciones	Que el recurso haya sido cargado correctamente.
Referencias	R16, R17, R18, R19, R20, R21, R22, R23, R24, R25.
Curso Normal de Eventos:	

Acción del Actor	Respuesta del Sistema
1-Solicita la encriptación del recurso	2- Se separa la información de los bloques y se aplica una función de conmutación entre sí a cada uno.
	3- Se almacena en el buffer los bloques encriptados.
	4- Se integra la información de los bloques para crear el nuevo fichero encriptado.
	5- Se libera el buffer. Se cierra el archivo.
Curso Alternativo de los eventos:	
	2.1- Si el recurso no fue cargado se muestra un mensaje de error.
Poscondiciones:	

Tabla 6: Expansión del Caso de Uso Encriptar Recurso

Nombre del Caso de Uso	Abrir Recurso Encriptado
Actores	Aplicación Final
Propósito	Abrir los recursos encriptados
Resumen: El caso de uso se inicia cuando la aplicación solicita abrir un recurso, luego de verificar que el mismo existe.	
Precondiciones	Que exista el recurso
Referencias	R26,R27
Curso Normal de Eventos:	
Acción del Actor	Respuesta del Sistema
1- Solicita abrir un recurso especificando la dirección	2- Se verifica que el recurso existe.
Curso Alternativo de los eventos:	
	2.1- Si el recurso no existe se muestra un mensaje de error.
Poscondiciones:	La información del recurso debe quedar almacenada.

Tabla 7: Expansión del Caso de Uso Abrir Recurso Encriptado

Nombre del Caso de Uso	Inicializar Funciones de Desencriptado
Actores	Aplicación Final
Propósito	Inicializar las funciones necesarias para realizar el desencriptado.
Resumen: El caso de uso se inicia cuando la aplicación final solicita la inicialización de las funciones necesarias para iniciar el proceso de desencriptación.	
Precondiciones	Que el recurso encriptado haya sido cargado.
Referencias	R28, R29, R30, R31, R32.
Curso Normal de Eventos:	
Acción del Actor	Respuesta del Sistema
1-Solicita la lectura de los datos del recurso encriptado.	2- Se verifica que el recurso encriptado se ha cargado correctamente.
	3- Se verifica que el fichero llave exista dentro del directorio.
	4- Se define el estado de los bloques de encriptación.
	5- Se carga la llave para crear las tablas de cifrado. Se verifica su integración y se inicializan las mismas.
Curso Alternativo de los eventos:	
	2.1- Si el recurso encriptado no fue cargado se muestra un mensaje de error.
	3.1- Si el fichero llave no existe se muestra un mensaje de error.
Poscondiciones:	

Tabla 8: Expansión del Caso de Uso Inicializar Funciones de Desencriptado

Nombre del Caso de Uso	Desencriptar Recurso
Actores	Aplicación Final
Propósito	Llevar a formato original el recurso encriptado.
Resumen: El caso de uso se inicia cuando la aplicación final solicita la desencriptación de los bloques encriptados para generar el archivo original.	
Precondiciones	Que el recurso encriptado haya sido cargado

	correctamente.
Referencias	R33, R34, R35, R36.
Curso Normal de Eventos:	
Acción del Actor	Respuesta del Sistema
1-Solicita la desencriptación del recurso.	2- Se separa la información de los bloques encriptados.
	3- Se aplica la función de desencriptado a los bloques. Se liberan los bloques.
	4- Se genera el archivo original. Se cierra el archivo.
Curso Alternativo de los eventos:	
Poscondiciones:	

Tabla 9: Expansión del Caso de Uso Desencriptar Recurso

Nombre del Caso de Uso	Integrar al motor gráfico OGRE
Actores	Aplicación Final
Propósito	Referenciar la dirección del recurso desencriptado al motor gráfico OGRE.
Resumen: El caso de uso se inicia cuando la aplicación final accede al fichero de configuración de OGRE y registra la dirección del archivo desencriptado.	
Precondiciones	Que se haya generado el recurso.
Referencias	R37
Curso Normal de Eventos:	
Acción del Actor	Respuesta del Sistema
1-Solicita la integración al motor gráfico.	2- Se accede al fichero de configuración del OGRE.
	3- Se realiza una búsqueda previa de la existencia de la dirección del recurso.
	4.- Se registra la dirección del nuevo recurso.
Curso Alternativo de los eventos:	
	3.1- Si la dirección ya se encuentra registrada se

	muestra un mensaje de error.
Poscondiciones:	

Tabla 10: Expansión del Caso de Uso Integrar al Motor Gráfico OGRE

CAPÍTULO #3. DISEÑO E IMPLEMENTACIÓN.

Introducción

En el presente capítulo se muestra el diseño de clases del sistema propuesto, donde se definen las responsabilidades de estas y sus relaciones. Además se muestran los diagramas de secuencia por casos de uso que intervendrán en el primer ciclo de desarrollo de software, facilitando el entendimiento de la aplicación.

Esta etapa del proyecto se conforma con el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .h y .cpp correspondientes a la implementación en C++. También se da a conocer la nomenclatura de las versiones y los estándares de codificación.

3.1- Diagrama de Diseño de Clases

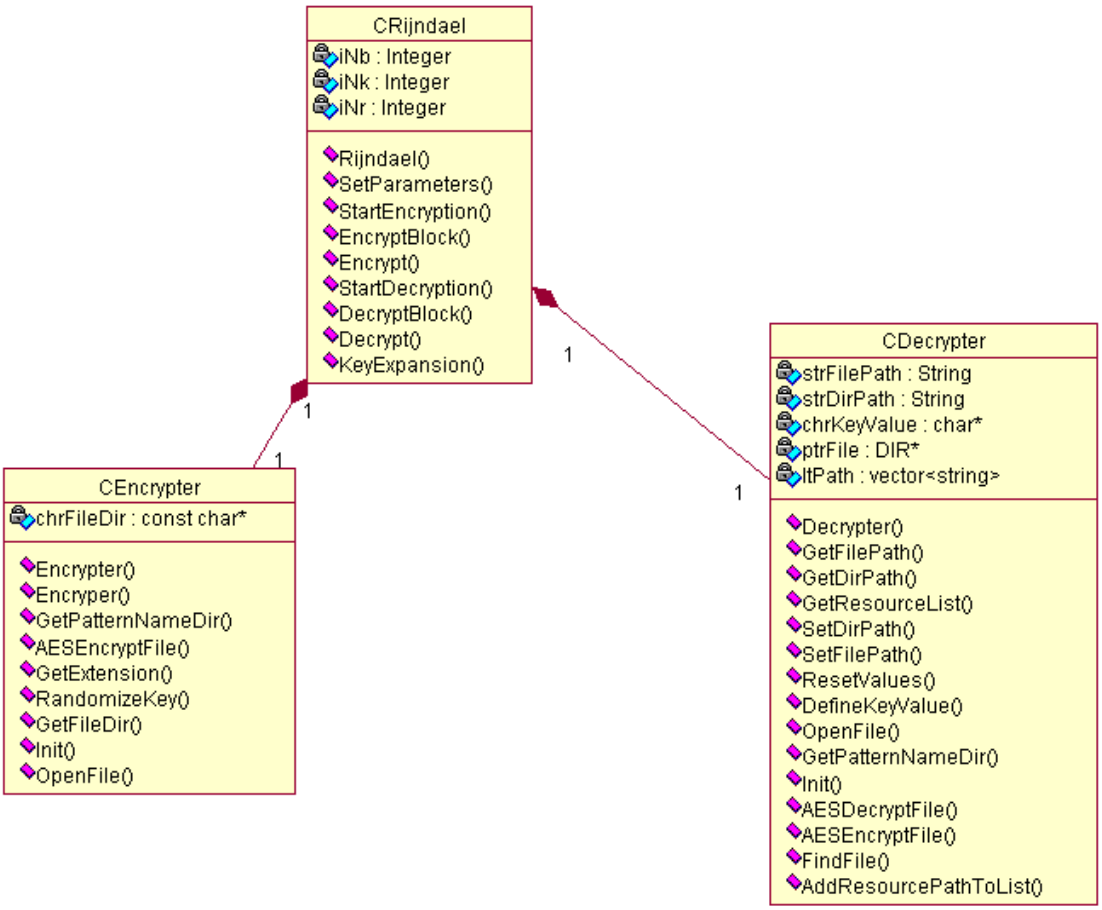

Figura 6: Diagrama de Diseño de Clases

El diagrama de diseño de clases que se propone, correspondiente a la Figura 6, muestra la composición existente entre las clases “CEncrypter” y “CDecrypter” con la clase “CRijndael”.

El algoritmo Rijndael va a ser el encargado de encriptar y desencriptar los recursos de la aplicación y para la confección del mismo se consideró la encriptación de los recursos que la aplicación vaya a utilizar, y a la hora de desencriptar se realizará desencriptando todo lo que fue encriptado anteriormente por tanto se define como mínimo de ocurrencia “1” para el caso de ambas clases CEncrypter y CDecrypter.

3.2- Descripción de las clases

Nombre: CRijndael	
Controladora	
Atributo	**Tipo**
iNb;	int
iNk;	int
iNr;	int
Para cada responsabilidad:	
Nombre:	SetParameters(int keylength, int blocklength = 128)
Descripción:	Se inicializan los tamaños de bloques con valores legales (128, 192 o 256).
Nombre:	Rijndael()
Descripción:	Constructor de la clase.
Nombre:	StartEncryption(const unsigned char * key)
Descripción:	Se inicializa el proceso de encriptación, convirtiendo y haciendo válida la llave de encriptación.
Nombre:	EncryptBlock(const unsigned char * datain, unsigned char * dataout)
Descripción:	Se realiza el proceso de encriptación de un simple bloque de dato. (128 bits por defecto).
Nombre:	Encrypt(const unsigned char * datain, unsigned char * dataout, unsigned long numBlocks, BlockMode mode = CBC)
Descripción:	Se inicia el proceso de encriptación de los bloques de datos, referenciando el modo más seguro.
Nombre:	StartDecryption(const unsigned char * key)

Descripción:	Se inicializa el proceso de desencriptación, convirtiendo y haciendo válida la llave de desencriptación, y se invierten las rondas para hacer más rápido el descifrado.
Nombre:	DecryptBlock(const unsigned char * datain, unsigned char * dataout)
Descripción:	Se realiza el proceso de desencriptación de un simple bloque de dato. (128 bits por defecto).
Nombre:	Decrypt(const unsigned char * datain, unsigned char * dataout, unsigned long numBlocks, BlockMode mode = CBC)
Descripción:	Se inicia el proceso de desencriptación de los bloques de datos, referenciando el modo más seguro.
Nombre:	KeyExpansion(const unsigned char * key)
Descripción:	Se manda a convertir la llave en formato válido y se realiza una copia local para el proceso de cifrado posterior.

Tabla 11: Descripción de la clase CRijndael

Nombre: CEncrypter	
Controladora	
Atributo	**Tipo**
chrFilePath;	const char *
Para cada responsabilidad:	
Nombre:	GetPatternNameDir()
Descripción:	Se manda a recorrer la dirección del archivo y se devuelve el nombre de la carpeta contenedora del fichero a procesar.

Nombre:	OpenFile(string strFileName)
Nombre:	Encryper()
Descripción:	Constructor de la clase.
Nombre:	Encrypter(const char * chrPathDir)
Descripción:	Se inicializa la dirección del directorio que contiene el fichero a procesar.
Descripción:	Se verifica si la dirección pasada por parámetro se puede abrir y acceder a ella correctamente.
Nombre:	AESEncryptFile(const char * chrFileName)
Descripción:	Se verifica la existencia del fichero de la llave de cifrado, en caso de no existir se crea con una llave randomizada. Se inicia el proceso de encriptación. Lanza un error si no encuentra archivos para encriptar. Se cierran todos los ficheros utilizados.
Nombre:	GetExtension(const char * chrName)
Descripción:	Se recorre todo el valor del nombre del archivo pasado por parámetro y se devuelve solamente la extensión del mismo.
Nombre:	GetFileDir()
Descripción:	Se devuelve la dirección del directorio.
Nombre:	Init()
Descripción:	Inicializa los parámetros de la llave de y los bloques de cifrado.
Nombre:	char * RandomizeKey()
Descripción:	Se genera aleatoria y randomizada una cadena de 24 caracteres que van a conformar la llave de cifrado con que se va a

	realizar el proceso de encriptación.

Tabla 12: Descripción de la clase CEncrypter

Nombre: CDecrypter	
Controladora	
Atributo	**Tipo**
strFilePath;	String
strDirPath;	String
chrKeyValue;	char*
ptrFile;	DIR *
ltPath;	vector<string>
Para cada responsabilidad:	
Nombre:	GetFilePath()
Descripción:	Se devuelve la dirección del fichero que se va a procesar.
Nombre:	Decrypter()
Descripción:	Constructor de la clase.
Nombre:	GetDirPath()
Descripción:	Se devuelve la dirección del directorio que contiene el fichero a procesar.
Nombre:	GetResourceList()
Descripción:	Se devuelve la lista que contiene las direcciones de los recursos que se van a utilizar.
Nombre:	SetDirPath(string strNewPath)
Descripción:	Se cambia por un nuevo valor la dirección del directorio que contiene el fichero a procesar.
Nombre:	SetFilePath(string strNewPath)
Descripción:	Se cambia por un nuevo valor la dirección

	del fichero a procesar.
Nombre:	ResetValues()
Descripción:	Se resetean los valores de los atributos.
Nombre:	DefineKeyValue(char * chrValue)
Descripción:	Se define el valor de la llave de cifrado con la que se va a procesar el fichero.
Nombre:	OpenFile(string strFileName)
Descripción:	Se verifica si la dirección pasada por parámetro se puede abrir y acceder a ella correctamente.
Nombre:	GetPatternNameDir()
Descripción:	Se manda a recorrer la dirección del archivo y se devuelve el nombre de la carpeta contenedora del fichero a procesar.
Nombre:	Init()
Descripción:	Inicializa los parámetros de la llave de y los bloques de cifrado.
Nombre:	AESDecryptFile(string strFileName)
Descripción:	Se verifica la existencia del fichero de la llave de cifrado válida correspondiente al directorio, en caso de existir, se inicia el proceso de desencriptación. Lanza un error si no encuentra el archivo pasado por parámetro a desencriptar. Se cierran todos los ficheros utilizados.
Nombre:	AESEncryptFile()
Descripción:	Se verifica la existencia del fichero de la llave de cifrado, en caso de no existir se crea con una llave randomizada. Se inicia el

	proceso de encriptación. Lanza un error si no encuentra archivos para encriptar. Se cierran todos los ficheros utilizados.
Nombre:	FindFile(string strFileName, eAction eMode)
Descripción:	Se busca en el directorio especificado la existencia del fichero y en caso de no encontrarlo se emite un mensaje de error.
Nombre:	AddResourcePathToList(string strPath)
Descripción:	Se verifica que la dirección pasada por parámetro es válida, y en caso de serlo se adiciona a la lista de direcciones de los recursos.

Tabla 13: Descripción de la clase CDecrypter

3.3- Diagramas de Secuencia

A continuación se ilustran los diagramas de secuencia vinculados a los Casos de Usos especificados**.**

El diagrama de secuencia “Abrir Recurso” el cual está representado en la Figura 7 muestra al actor Aplicación Encriptar interactuando con la clase CEncrypter.

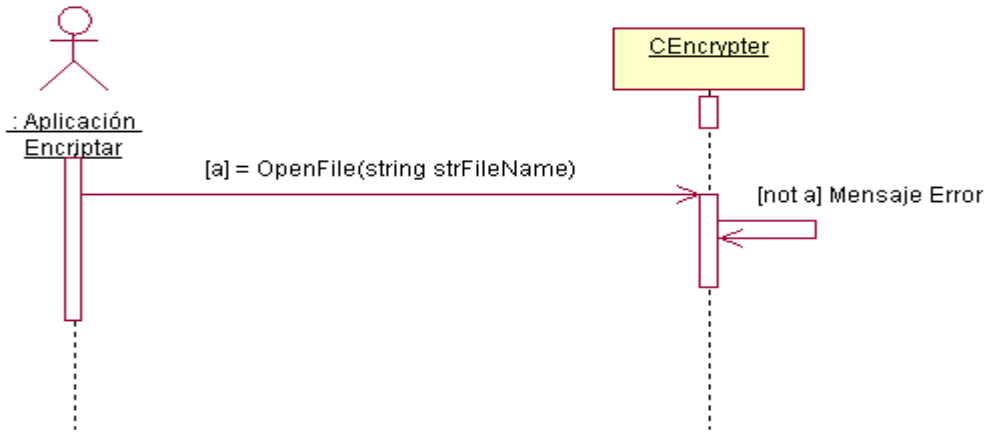

Figura 7: Diagrama de Secuencia Abrir Recurso de la Aplicación Encriptar

El diagrama de secuencia “Inicializar Funciones de Encriptación” el cual está representado en la Figura 8 muestra al actor Aplicación Encriptar interactuando con la clase CEncrypter y con la clase CRijndael, la cual envía, cuando el actor desee inicializar las funciones de encriptación, el mensaje de inicializar estas funciones a las clases encargadas de manipular esta información y como resultado se inicializan todas las funciones para inicializar el proceso.

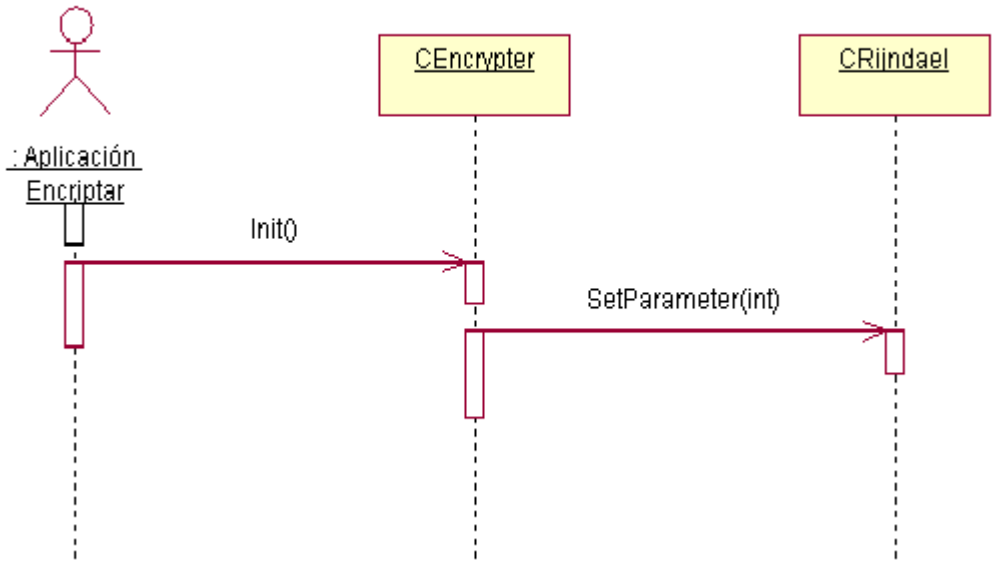

Figura 8: Diagrama de Secuencia Inicializar Funciones de Encriptación de la Aplicación Encriptar

El diagrama de secuencia “Encriptar Recursos” el cual está representado en la Figura 9 muestra al actor Aplicación Encriptar interactuando con la clase CEncrypter y con la clase CRijndael, la cual envía a la clase encargada de manipular la información referente al proceso de encriptación de datos, el mensaje de comenzar con el proceso de encriptación. La clase que controla este proceso es la clase CEncrypter, la cual carga en memoria todos los archivos a encriptar comenzando así este proceso.

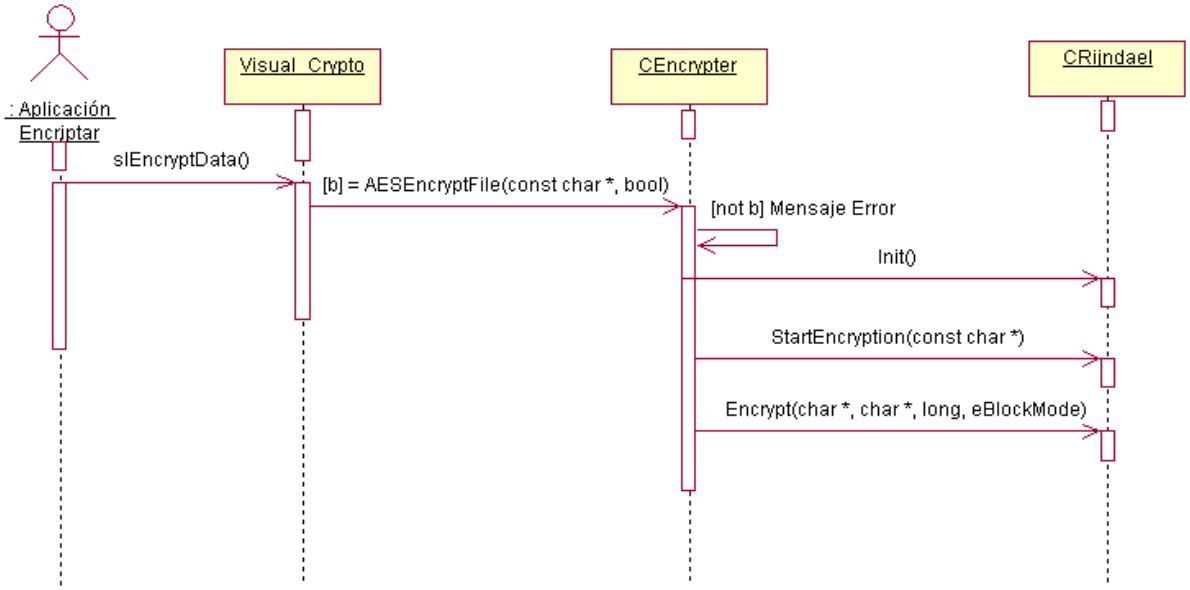

Figura 9: Diagrama de Secuencia Encriptar Recurso de la Aplicación Encriptar

El diagrama de secuencia “Abrir Recurso Encriptado” el cual está representado en la Figura 10 muestra al actor Aplicación Final interactuando con la clase CDecrypter.

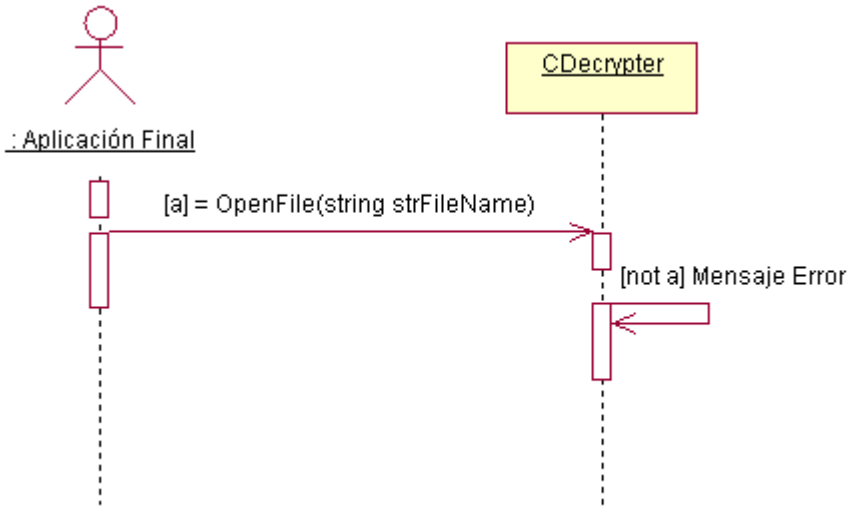

Figura 10: Diagrama de Secuencia Abrir Recurso Encriptado de la Aplicación Final

El diagrama de secuencia “Inicializar Funciones de Desencriptación” el cual está representado en la Figura 11 muestra al actor Aplicación Final interactuando con la clase CDecrypter y con la clase CRijndael, la cual envía, cuando el actor desee inicializar las funciones de desencriptación, el mensaje de inicializar estas funciones a las clases encargadas de manipular esta información y como resultado se inicializan todas las funciones para inicializar la desencriptación.

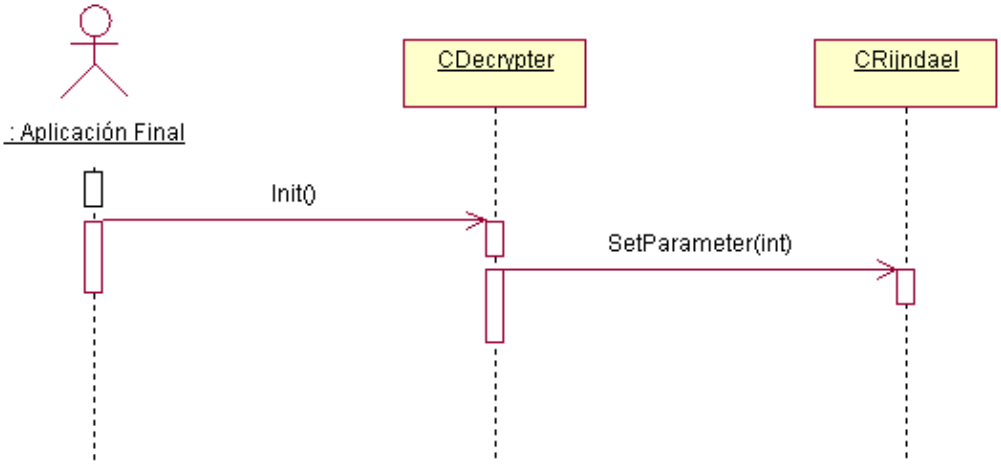

Figura 11: Diagrama de Secuencia Inicializar Funciones de Desencriptación de la Aplicación Final

El diagrama de secuencia “Desencriptar Recursos” el cual está representado en la Figura 12 muestra al actor Aplicación Final interactuando con la clase CDecrypter y con la clase CRijndael, la cual envía a la clase encargada de manipular la información referente al proceso de desencriptación de datos, el mensaje de comenzar con el proceso de desencriptación. La clase que controla este proceso es la clase CDecrypter, la cual carga en memoria todos los archivos a desencriptar comenzando así este proceso.

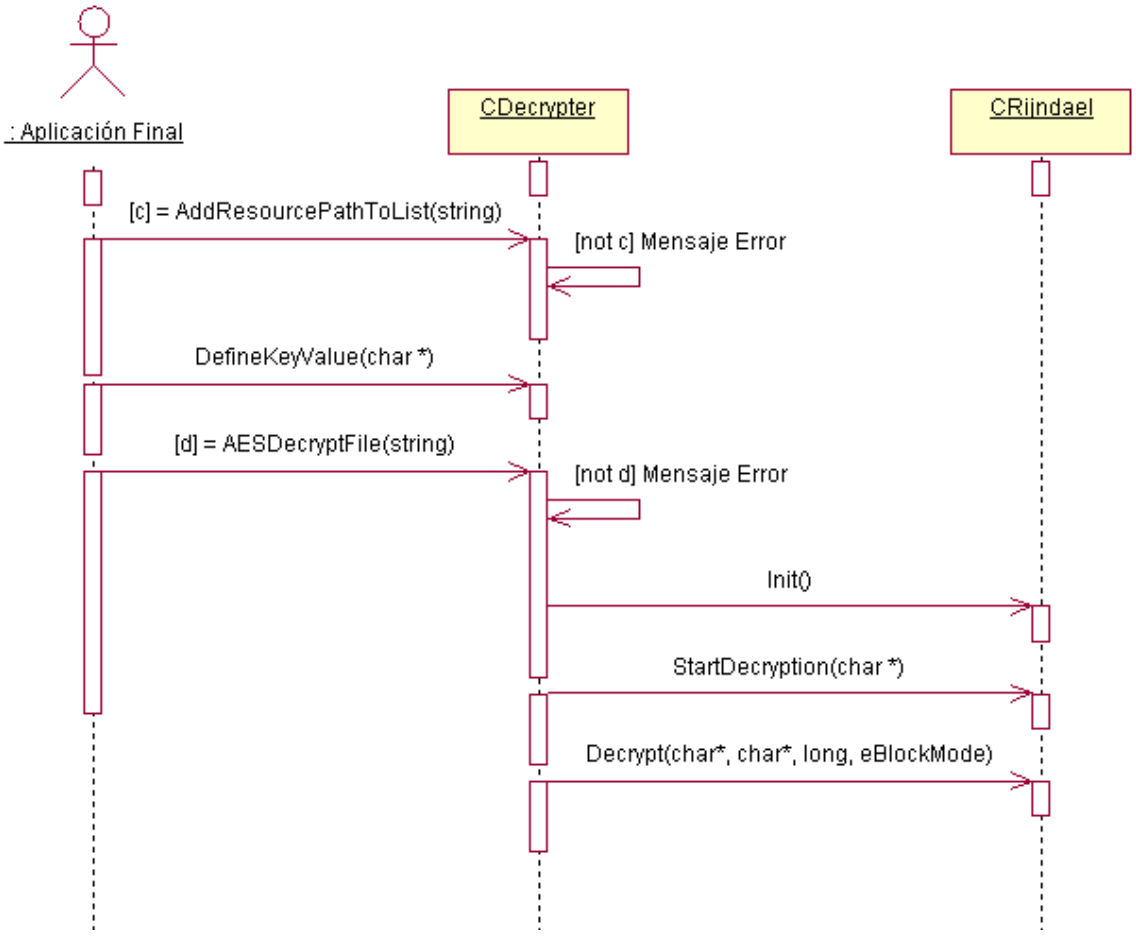

Figura 12: Diagrama de Secuencia Desencriptar Recurso de la Aplicación Final

3.4- Estándares de Codificación

El código del software implementado sigue algunos estándares propuestos por el equipo de desarrollo, respetando los estándares de codificación del lenguaje con el cual se programó. Está programado en inglés, debido a que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.

El conocimiento de los estándares seguidos para el desarrollo de la misma permitirá un mayor entendimiento del código.

Nombre de los ficheros:

Los nombres de los ficheros .h y .cpp siempre se nombran con letra mayúscula.
Ej: Unit.h
Unit.cpp

Enumerados:

Para los enumerados se utiliza el indicador “e” para el encabezado del nombre del tipo, y las constantes van en mayúsculas.
Ej: enum eMyEnum
{
VALUE = 0;
};

Estructuras:

Se utiliza el indicador “st” para indicar que es una estructura. El nombre de la estructura comienza con el indicador, seguido del nombre comenzando con letra mayúscula.
Ej: struct type stName = NULL;

Clases:

Se utiliza el indicador “C” para indicar que es una clase. Ver más adelante la nomenclatura de las variables miembros.
Ej: class CClassName;

Declaración de variables:

Los nombres de las variables comienzan con un identificador del tipo de dato al que correspondan, como se muestra a continuación.

Tipos simples:

Ej:
string strName;
int iName;
bool blName;
char* chrName;

Instancias de tipos creados:

Ej:
CClassName ptrName;

Métodos:

En el caso de los métodos, siempre comienzan con letra mayúscula, en caso de estar formados por más de una palabra, cada una de ellas comienzan con letra mayúscula sin dejar espacios entre ellas.
Ej: bool MyMethod();

Constructor y destructor:

Ej: CClassName (bool bl);
~CClassName ();

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases se nombrarán “Get” o “Set”, seguido del nombre de la variable a la que se accede sin antecederle el tipo de dato de la variable.
Ej: Para la siguiente variable, los métodos de acceso serían:
int iVar; //variable
int GetVar();
void SetVar(int iNewVar);

3.5- Convenciones de Programación

3.5.1- Con respecto a la INDENTACION

- Usar cuatro espacios como unidad de indentación.

- Cuando una expresión no entre en una sola línea, se debe romper de acuerdo a estos principios generales:

  - Romper después de una coma.
  - Romper antes de un operador.
  - Alinear la nueva línea al principio de la expresión al mismo nivel de la línea anterior.
  - Si las reglas anteriores conducen a la confusión del código o el código se alinea contra el margen derecho, indentamos menos espacios.

Ej:

```
SomeMethod(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

int iVar = someMethod1(longExpression1,
                someMethod2(longExpression2,
                        longExpression3));
```

3.5.2- Con respecto a las DECLARACIONES

- Se debe declarar cada variable en su propia línea.
- Es válido inicializar variables al momento de su declaración.

Ejemplos válidos:

```
int iVar = 9;
int iCant;
```

NO utilizar:

```
int iVar, iCant;
```

3.5.3- Con respecto a las SENTENCIAS

- Buscar usar sentencias simples, cada sentencia debe ser escrita en su propia línea, es decir, no utilizar líneas como:

```
iVar++; a = b + 2;
```

3.5.4- Sentencias de retorno

- Una sentencia return con un valor no debe usar paréntesis a menos que hagan el valor de retorno más obvio de alguna manera.

  Ejemplo:

```
return;
return miDiscoDuro.size();
return (tamanyo ? tamanyo : tamanyoPorDefecto);
```

3.5.5- Con respecto a las SENTENCIAS IF

- Deben tener la siguiente forma:

```
if (condition)
    statements;

if (condition)
 {
    statements;
 }
else
{
    statements;
}
```

3.5.6- Con respecto a las SENTENCIAS FOR

- Deben tener la siguiente forma:

```
for (initialization; condition; update)
{
        statements;
}
```

3.5.7- Con respecto a las SENTENCIAS WHILE

- Deben tener la siguiente forma:

```
while (condition)
{
        statements;
}
```

- Una sentencia while vacía debería tener la siguiente forma:

```
while (condition);
```

3.5.8- Con respecto a las SENTENCIAS SWITCH

- Deben tener la siguiente forma:

```
switch (condition)
{
case ABC:
        statements;
        /* no se concreta */
case DEF:
        statements;
        break;
case XYZ:
        statements;
        break;
default:
      statements;
}
```

- Cada vez que un caso no se concreta (no incluir la sentencia break), agregue un comentario donde la sentencia break debería ir normalmente.
- Esto es mostrado en el ejemplo precedente con el comentario /* no se concreta */.
- Cada sentencia switch debe incluir un case por default.

3.6- Diagrama de Componentes

Se propone el diagrama de componente perteneciente a la Figura 13, para lograr un algoritmo multiplataforma. Existe similitud entre los métodos que implementa cada clase. Se utiliza una .cpp por cada .h en el caso de las clases.

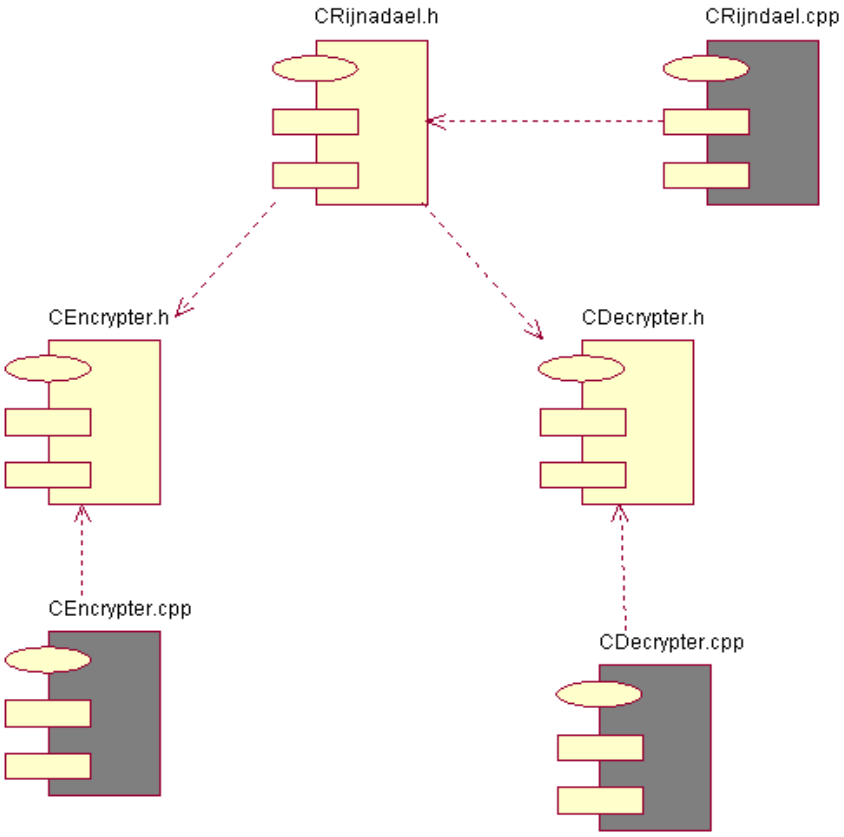

Figura 13: Diagrama de Componentes

3.7- Diagrama de Despliegue

Se propone el siguiente diagrama de despliegue perteneciente a la Figura 14 para tener una representación clara del nodo físico existente, teniendo en cuenta que todo el proceso de encriptación y desencriptación se lleva a cabo en la misma máquina donde estará corriendo la aplicación.

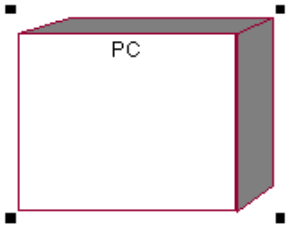

Figura 14: Diagrama de Despliegue

CONCLUSIONES

Al término de la presente investigación:

- ✓ Se implementó una aplicación capaz de darle protección a los recursos multimedia empleados en aplicaciones gráficas.
- ✓ Se mostró de forma funcional la utilización de la técnica alcanzada para la protección de los recursos multimedias utilizados en una aplicación desarrollada con el motor de render Ogre.
- ✓ La utilización del algoritmo Rijndael para encriptar los recursos multimedia es una solución factible que provee una mayor seguridad ante la piratería informática.

RECOMENDACIONES

Para el futuro trabajo con esta aplicación y dado la utilidad de la misma se recomienda:

- ✓ La utilización de este proceso de cifrado en otros motores gráficos, para el aseguramiento de sus recursos.
- ✓ Continuar desarrollando esta aplicación para que en un futuro se pueda integrar como un plugin al motor de Render Ogre.

BIBLIOGRAFIA REFERENCIADA

[1] Cifrado y Firmas Digitales, Copyright © 1990-2010 Zator Systems, publicado por A.J. Millán ajm@zator.com, en Julio de 1998 y su última actualización fue realizada el 9 de abril de 2010. http://www.zator.com/Internet/A6_4.htm

[2] Cifrado XOR
http://fabbc.awardspace.com/Cifrado-XOR.php

[3] Criptografía y la Informática El Vortex Tecnologic, Publicado en Uncategorized con etiquetas Freedom Day en Septiembre 15, 2009 por Raúl Espinola. http://raulespinola.wordpress.com/2009/03/27/criptografia-y-la-informatica/

[4] Historia de la criptografía, publicado por Steven Levy, Cripto: Cómo los informáticos libertarios vencieron al gobierno y salvaguardaron la intimidad, Madrid, Alianza, 2002.
http://wapedia.mobi/es/Historia_de_la_criptograf%C3%ADa

[5] Marcos Criptografía: Algoritmos Simétricos-Fly and Enjoy.Net, Marcos del Pozo [Blog].
http://developersdotnet.com/blogs/marcos/default.aspx

[6] Seguridad Informática/Criptología – Algoritmos Simétricos
http:www.segu.info.com.ar/criptología/simétricos.html

[7] El Gamal, A.Ribagorda, Glosario de Términos de Seguridad de las T.I, Ediciones CODA, 1997.
https://www.ccn-cert.cni.es/publico/serieCCN-STIC401/es/e/elgamal.htm

[8] Criptografía y seguridad en computadores (3ra ed). Manuel José Lucena López.
http://www.themalia.es/admin/img/documentos/200506281022060.Criptografia.pdf

[9] **Delgado Vera, Palacios Rafael.** Introducción a la criptografía. Tipos de algoritmos. Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas.

[10] La Encriptación
http://www.alipso.com/monografias4/encriptación/

[11] Muñoz Muñoz, Alfonso. Septiembre 2004 Madrid. Seguridad Europea para EEUU. Algoritmo Criptográfico Rijndael.

[12] AES page available vía, Publicado por National Institute of Standars and Technologic. http://www.nist.gov/CryptoToolkit.

[13] Computer Security Objects Register (CSOR), Publicado por National Institute of Standars and Technologic.
http://csrc.nist.gov/csor/.

[14] J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 1999, available at.

[15] J. Daemen and V. Rijmen, The block cipher Rijndael, Smart Card research and Applications, LNCS 1820, Springer-Verlag, pp. 288-296.

[16] J. Nechvatal, et. al., Report on the Development of the Advanced Encryption Standard (AES), National Institute of Standards and Technology, October 2, 2000, available at.

[17] Algoritmos Simétricos de Cifrado de Claves que utiliza BlackBerry, Publicado por BlackBerry Enterprise Server per IBM Lotus Domino.
http://docs.blackberry.com/es/admin/deliverables/12854/Standard_encryption_algorithm_that_BESoln_uses_193578_11.jsp

[18] El Blog de Daniel Lerch: DES y TRIPLE DES, Publicado por Daniel Lerch, 9 de Junio de 2007.
http://dlerch.blogspot.com/2007/06/des-y-triple-des.html

[19] Pagina Nueva 1 – Eurologic SA
http://www.eurologic.es/cifrado/simetric.html

[20] Seguridad-Algoritmos de Codificación Fuerte, Publicado por PentaWare Inc. compañía distribuidora de Software en el año 2005.
http://www.pentaware.com/PW_es/PentaSuite_Security.htm

[21] Algoritmos DES, 3DES, RC4 e IDEA/Kriptólisis, Publicado por Pepo el 12 de Noviembre de 2005. http://www.kriptopolis.org/algoritmos-des-3des-rc4-e-idea

[22] Seguridad de la información, Copyright © Cristian Borghello 2000 – 2009. http://www.segu-info.com.ar/proyectos/p1_algoritmos-llave-privada.htm

[23] Lista de algoritmos de encriptación http://comunidad.dragonjar.org/f156/lista-de-algoritmos-de-encriptacion-8807/

[24] Introducción a Rational Rose, Publicado por Rubén González Blanco y Sergio Pérez Tobalina del Departamento de Ingeniería y Sistemas Informáticos de la Universidad de Catalunya. http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAYQFjAA&url=http%3A%2F%2Fwww.essi.upc.edu%2F~ese%2Fweb%2Fdocuments%2Flab%2F0304Q2%2Flessons%2Flese2%2FLESE2%2520%2520Introduccion%2520a%2520Rational%2520Rose.ppt&rct=j&q=rational+rose+caracteristicas&ei=a9S4fUFIL7lweY15SHBw&usg=AFQjCNEz64ZXV5AtmvkH1QHjN5pJT-mhdA

[25] ¿Por qué OGRE?, (Cortizo, 2007) J.C. Cortizo Pérez: *Motores 3D*, Asignatura Motores del Master de la Universidad Europea de Madrid. http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAYQFjAA&url=http%3A%2F%2Ftamudo84.googlepages.com%2FIntroduccinaOgre-PorquOGRE.pdf&rct=j&q=caracteristicas+y+ventajas+de+ogre&ei=UPS9S8caxP-WB5iblKoH&usg=AFQjCNHz-oMRX-4Pwg_WCJGb4z9qQ-cSsQ

[26] (OGRE3D, 2007) Ogre3D.org: *What Is OGRE,* http://www.ogre3d.org/ *2007*.

[27] Microsoft Visual Studio 2008-Programación-ISV-7 Abril 2008, Publicado por Boadilla del Monte el lunes 5 de mayo de 2005. http://www.mkm-pi.com/mkmpi.php?article2155

[28] Angel Angel José de Jesús, AES - Advanced Encryption Standard; Noviembre de 2005; jjaa_@hotmail.com; pp. 89. http://computacion.cs.cinvestav.mx/~jjangel/aes/AES_v2005_jjaa.pdf

[29] A.E.S. El algoritmo de encriptación del próximo siglo, Publicado por Ing. Mónica Liberatori en el 2005. http://www3.fi.mdp.edu.ar/electronica/articulos/Criptografia.doc

[30] Daemen, Joan; Rijmen, Vicent; The Rijndael Block Cipher A.E.S. Proposal.

[31] Federal Information Processing Standards Publication 197. Specification for the Advanced Encryption Standard (A.E.S).

[32] Maizel. Patricia V.; "Implementación del algoritmo A.E.S. (Advanced EncryptionStandard)", Ingeniería Electrónica; 2004; Informe técnico.

[33] Muñoz Muñoz. Alfonso; CRIPTOSISTEMA RIJNDAEL, El Secreto de las Comunicaciones, Copyright 2007. Pp. 47.

[34] Seguridad y algoritmos de encriptación; *Copyright © 2001-2008* Ranquel Technologies; www.cryptoforge.com

[35] Qt Designer – Monografías.com. Escrito por Rolando Godoy Lara y Cristhian Castillo Martínez, estudiantes de la Facultad de Ciencias de la Computación y Electrónica de la Universidad Tecnológica América de la Ciudad de Quito, Ecuador.
http://www.monografias.com/trabajos15/qt-designer/qt-designer.shtml

GLOSARIO

API: Interfaz de programación de aplicaciones (Applications Programming Interface): una serie de funciones que están disponibles para realizar programas para un cierto entorno.

Campos de Galois: Cuerpo que contiene un número finito de elementos.

Cifrado: El cifrado es un método que permite aumentar la seguridad de un mensaje o de un archivo mediante la codificación del contenido, de manera que sólo pueda leerlo la persona que cuente con la clave de cifrado adecuada para descodificarlo.

C++: Lenguaje de programación orientado a objetos.

Desencriptar: En Informática, descodificar mediante las claves adecuadas la información que previamente se había encriptado.

Encriptar: Técnica por la que la información se hace ilegible para terceras personas. Para poder acceder a ella es necesaria una clave que sólo conocen el emisor y el receptor. Se usa para evitar el robo de información.

Framework: Es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación.

GUI: Significa graphical user interface, en castellano interfaz gráfica de usuario. Permite a los usuarios navegar e interactuar con las informaciones en la pantalla de su ordenador utilizando un mouse para señalar, pulsar y desplazar iconos y otros datos de la pantalla, en lugar de escribir palabras y frases. El World Wide Web es un ejemplo de un GUI diseñado para facilitar la navegación en Internet.

Multiplataforma: Que la aplicación corra en varios sistemas operativos.

OGRE: Significa Object Oriented Graphics Engine, lo que en castellano significa motor gráfico orientado a objetos.

OpenGl: Es una biblioteca gráfica desarrollada originalmente por Silicon Graphics Incorporated (SGI). OpenGL significa Open Graphics Library, cuya traducción es biblioteca abierta de gráficos.

Realidad Virtual: La Realidad Virtual es simulación por computadora, dinámica y tridimensional, con alto contenido gráfico, acústico y táctil, orientada a la visualización de situaciones y variables complejas, durante la cual el usuario ingresa, a través del uso de sofisticados dispositivos de entrada, a "mundos" que aparentan ser reales, resultando inmerso en ambientes altamente participativos, de origen artificial.