# Aplicación INERCIPHER

Trabajo Final de Carrera

***Cifrado de archivos***

Nombre de Estudiante ***Inga Ermolova Ermolova***
***ETIS***

Nombre de Consultor ***Antoni Martínez Balleste***

**Data de presentación** **10.01.2006**

## Agradecimientos.

*Este proyecto nunca se hubiera hecho realidad sin el apoyo, consejos y ayuda de todos mis profesores. Quiero dar gracias a todas las personas junto con quienes tanto he aprendido.*

*A Joan, por la paciencia, el apoyo y la enorme confianza en mi.*


## Resumen.


El programa INERCIPHER es una aplicación para realizar el cifrado y descifrado de los archivos y para verificar la integridad de los documentos. Como base para desarrollar estas funcionalidades han sido utilizados dos tipos de cifrado de clave compartida: cifrado en flujo y cifrado en bloque.

Cifrado en flujo se emplea para conseguir una secuencia pseudoaleatoria a partir de la clave secreta, que posteriormente se utiliza como secuencia de cifrado. Para conseguir este propósito ha sido implementado el generador de las secuencias pseudoaleatorias – generador de Geffe.

Para realizar cifrado/descifrado ha sido utilizado el algoritmo AES (Advanced Encription Estandard), que se considera el algoritmo de cifrado de siglo XXI (este estándar puede utilizarse hasta el año 2060), con la clave de 128 bits. AES es uno de los métodos de cifrado en bloque y ha sido utilizado el modo de cifrado CBC dado que en la utilización de la aplicación la tasa de errores producidos será baja.

Para realizar la verificación de integridad de documentos ha sido creada la parte de programa que permite generar el código de autenticación MAC, que posteriormente se utiliza para la comprobación de integridad de documento, implementando el algoritmo AesXCbcMac, presentado en septiembre de 2003 por S.Frankel (NIST, National Institute of Standarts and Technology) y H.Herbert (Intel).

Finalmente ha sido construida una interfaz gráfica que permite la utilización cómoda y fácil de la aplicación.

En la Memoria presente se explican la base teórica de cada tipo de cifrado utilizado, la implementación de programa, las pruebas de código realizados y se adjunta el Manual de usuario ilustrado con las imágenes de pantalla correspondiente a la ejecución de aplicación INERCIPHER.

# INDEX.

**CAPÍTULO 1 ..... 4**

1.1 Introducción **..... 4**
1.2 Objetivos del proyecto ..... 4
1.3 Enfoque del proyecto ..... 5
1.4 Planificación del proyecto ..... 5
1.5 Estructura general de aplicación ..... 5

**CAPÍTULO 2**.
Generador de clave compartida..... 8

2.1 Cifrados de flujo ..... 8
2.2 LFSR ..... 8
2.3 Generador de Geffe ..... 10

**CAPÍTULO 3.**
Implementación de secuencia pseudoaleatoria
en INERCIPHER **.....12**

3.1 Implementación de bloque Generador de Geffe ..... 12
3.2 Pruebas de funcionamiento de generador de Geffe ..... 13

**CAPÍTULO 4.**
Bloque Cifrador/Descifrador **..... 15**

4.1 Cifrado en bloque ..... 15
4.2 Cifrado AES ..... 16
4.3 Esquema de cifrado AES ..... 17
4.4 Esquema de clave en AES ..... 18
4.5 Modo de implementación de AES
en bloque Cifrador/Descifrador .....19
4.6 Implementación de bloque Cifrador/Descifrador .....21

**CAPÍTULO 5**
Bloque DocIntegrity **..... 23**

5.1 Código de integridad MAC ..... 23

5.2 Algoritmo AesXCbc ..... 23
5.3 Implementación de bloque DocIntegrity ..... 24
5.4 Pruebas de funcionamiento de AesXCbc ..... 25

**Capítulo 6.**
Interfaz gráfica ..... 27

**Capítulo 7.**
Conclusiones .....28

**Glosario ..... 29**

**Bibliografía ..... 30**

**Anexo A. ..... 32**

**Anexo B.**
**Creación de vector inicial de la contraseña ..... 33**

**Anexo C.**
**Manual de usuario ..... 34**

## Lista de figuras.

***Figura 1.*** Esquema general de aplicación INERCIPHER ..... 6
***Figura 2.*** Diagrama de casos de uso de programa INERCIFER ..... 7
***Figura 3.*** Esquema de LFSR ..... 8
***Figura 4.*** Esquema de generador de Geffe .....10
***Figura 5.*** Distribución de bits de contraseña ..... 12
***Figura 6.*** Esquema general de bloque funcional Geffe ..... 13
***Figura 7.*** Captura de la pantalla al realizar las pruebas de generador de Geffe ..... 14
***Figura 8.*** Esquema general de AES .....17
***Figura 9.*** Esquema de expansión de clave en AES ..... 19
***Figura 10.*** Esquema de modo de cifrado CBC ..... 20
***Figura 11.*** Diagrama de clases de bloque Cifrador/Descifrador ..... 21
***Figura 12.*** Diagrama de clases de bloque DocIntegrity..... 24
***Figura 13.*** Captura de la pantalla al realizar las pruebas con vectores de teste ..... 26
***Figura 14.*** Ejecución de INERCIPHER con interfaz gráfica ..... 27

# Capítulo 1.

## 1.1 INTRODUCCIÓN.

La información ha sido un activo fundamental en los negocios desde la antigüedad, cumpliendo un rol primordial en el éxito o fracaso de los mismos.

En el entorno competitivo actual, las empresas cada vez más dependen de la disponibilidad y seguridad de la información estratégica para actuar en forma adecuada. Se necesita que esta información esté disponible siempre, sólo para las personas indicadas, y que se pueda asegurar su integridad.

Tradicionalmente, las empresas han enfocado sus esfuerzos de seguridad a protegerse frente a amenazas externas. Estos esfuerzos son necesarios, en especial desde que el uso de Internet a nivel corporativo aumentó, incluyendo la publicación de servicios que acceden a servidores que una vez fueron de uso exclusivamente interno. Los firewalls y detectores de intrusos proveen una excelente línea de defensa frente a estas amenazas, permitiendo a los administradores definir que usos legítimos pueden hacer los usuarios de los recursos de la empresa.

Sin embargo, los atacantes externos no son la única amenaza para las organizaciones. Diversos estudios indican que en forma creciente, las organizaciones resultan víctimas de ataques originados desde el interior. Para el 2005, el 60% de las violaciones de seguridad en las empresas tendrán motivaciones políticas o económicas, y serán el resultado de la acción de empleados solos o en conjunción con conspiradores externos (Gartner). El 49% de las empresas se ven afectadas por ataques internos (CSO Online). El 22 % ha sufrido sabotaje electrónico y hurto de información por parte de empleados CSO Online).[26]

Los atacantes internos cuentan con muchas ventajas: conocen el funcionamiento de la organización desde adentro, tienen algunos privilegios que necesitan para realizar su trabajo, y mayores posibilidades de conocer los puntos débiles de los sistemas.

Para lograr un nivel máximo de la seguridad de información, resulta fundamental que las organizaciones no solamente se protegen el acceso a la información crítica por parte de los extraños, también han de tomar en cuenta los riesgos internos.

## 1.2 OBJETIVOS DEL PROYECTO.

Los objetivos del proyecto INERCIPHER consiste en crear una herramienta que permitiría a los usuarios cifrar y descifrar los documentos críticos. Al guardar la información sensible en forma cifrada, se consigue la protección de esta información de los usuarios no autorizados sin implantar las restricciones de acceso excesivas.

Además la aplicación incluye la opción de generar el código de integridad MAC para aquellos documentos que contienen la información con permiso de lectura para varios grupos de usuarios, por lo cual no es necesario cifrarlos, pero esta información es crítica desde punto de vista de modificaciones no autorizados.

## 1.3 ENFOQUE DEL PROYECTO.

Para poder realizar el cifrado de los documentos en primer lugar hay que elegir un método de cifrado adecuado para cumplir los objetivos de la aplicación y además este método ha de ser seguro y rápido y que necesite mínimo de recursos del sistema. Para generar los documentos cifrados a partir de los documentos con texto claro para guardarlos posteriormente en el los dispositivos de memoria de un PC ha sido elegido el cifrado de clave compartida. De ese modo el cifrado y el descifrado de los documentos se realiza utilizando la misma clave que se genera a partir de la contraseña del usuario. Para la comprobación de la integridad de un documento también se utiliza el mismo tipo de cifrado: se genera el código MAC de un documento utilizando la clave generada a partir de la contraseña del usuario y esta misma contraseña ha de ser utilizada para verificación de la integridad de documento.

Para implementación de la aplicación ha sido utilizado el lenguaje de programación Java, la librería estándar de Java y la información publicada en la página oficial de IETF (*Internet Engineering Task Force* ) [17].

El hecho de no emplear las librerías criptográficas especiales como IAIK, CRYPTLIB, CRYPTIX, etc. facilita la utilización de la aplicación y limita los recursos necesarios a la instalación de VM de Java que ya va incluida en muchos sistemas operativos.

Para facilitar el uso de la aplicación y permitir el modo de ejecución interactivo ha sido creada la interfaz gráfica que ayuda al usuario seguir el proceso paso a paso.

## 1.4 PLANIFICACIÓN DEL PROYECTO.

En primer lugar ha sido recopilada y procesada la información relacionada con el cifrado de clave compartida: los métodos de cifrado, las ventajas y desventajas de cada método en particular, las posibilidades de cpriptoanálisis de los textos cifrados según el método de cifrado. Posteriormente, al elegir los métodos que serían utilizados, ha sido generado un esquema general del proyecto. Basándose en el esquema de aplicación han sido implementados los bloques funcionales del programa (generado el código-fuente de los bloques que permitirían cifrar/descifrar los archivos, generar el código MAC, verificar la integridad de documento). Las pruebas de funcionamiento correcto del programa se realizaban para cada bloque funcional por separado al acabar la implementación. Finalmente ha sido creada la interfaz gráfica que permite la interacción del usuario con la aplicación de la manera cómoda y clara (la utilización de la aplicación INERCIPHER no requiere la preparación especial previa del usuario).

## 1.5 ESTRUCTURA GENERAL DE APLICACIÓN.

Como resultado de realización del proyecto ha sido creada la aplicación **INERCIPHER*** .

Este programa no necesita la instalación y es muy fácil de utilizar gracias a su modo interactivo.

La aplicación consiste de tres bloques principales:

- Generador de la secuencia pseodoaleatoria (generador Geffe)
- Bloque cifrador/descifrador, que utiliza la secuencia pseudoaleatoria generada anteriormente como la clave de cifrado/descifrado.
- Bloque generador de código MAC/verificador (bloque DocIntegrity); para crear el código de integridad también es preciso disponer de la clave (secuencia pseudoaleatoria generada por generador de Geffe).

**ESQUEMA DE APLICACIÓN INERCIPHER.**

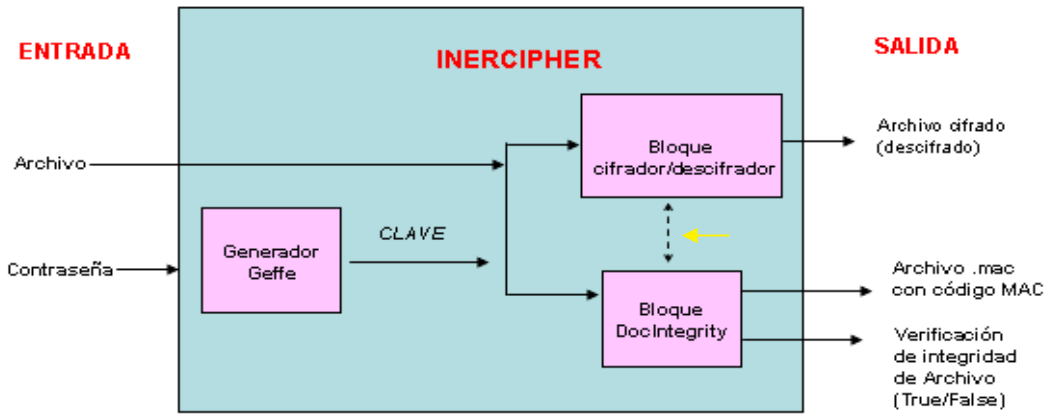


*Figura 1. Esquema general de aplicación INERCIPHER*

El bloque *DocIntegrity* y bloque cifrador/descifrador comparten una parte de código (algunas clases se utilizan para ambos bloques) para evitar la redundancia.

Utilizando el diagrama de caso de uso el programa INERCIPHER se puede presentar de manera siguiente:

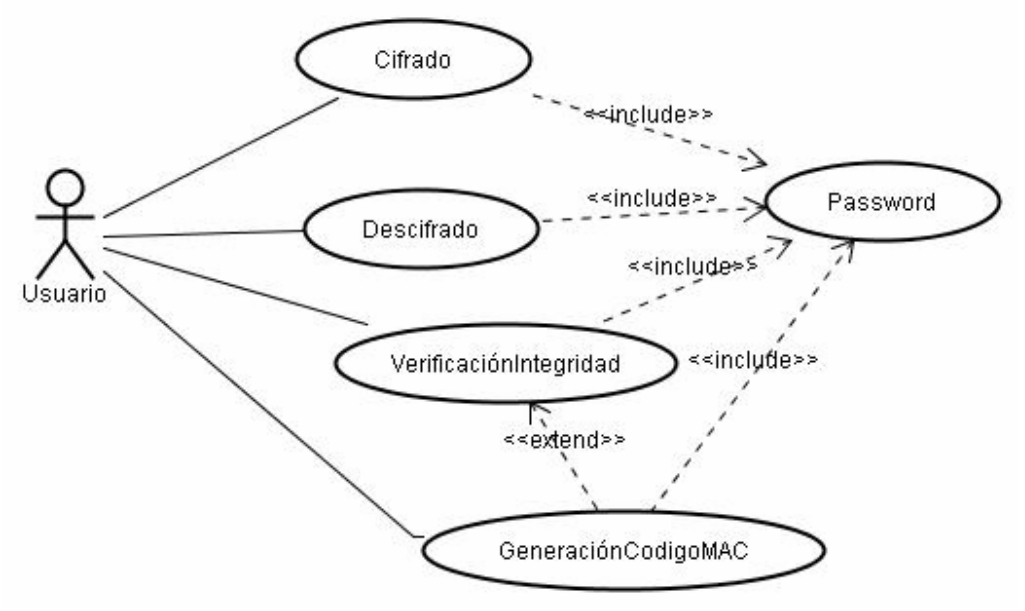


*Figura 2. Diagrama casos de uso de programa INERCIPHER.*

Para implementar las funcionalidades de la aplicación han sido utilizados dos tipos de cifrado de clave compartida: **cifrado de flujo** para generación de la clave secreta a partir de la contraseña de usuario y **cifrado en bloque** para cifrado de los archivos y la generación del código de integridad MAC.

En los capítulos siguientes se realiza la descripción detallada de cada bloque y justificación de la utilización de cada tipo de cifrado basada en las propiedades de cada algoritmo en particular y posibilidades de su implementación (recursos, velocidad de funcionamiento, necesidad de las herramientas y librerías especiales, etc.)

*Antes de elegir el nombre de la aplicación como definitivo ha sido realizadas las consultas necesarias para comprobar que este nombre no está patentado por otros autores anteriormente.

# Capítulo 2.

## GENERADOR de CLAVE COMPARTIDA

## 2.1 Cifrado de flujo.

Para conseguir un criptosistema incondicionalmente seguro se necesita como mínimo tantos bits de la clave como contiene el texto para cifrar. El único que consigue está seguridad incondicional es el cifrado de Vernam [Anexo A], pero es muy ineficiente a la hora de realizar cifrado.

Los **cifrados de flujo** surgen como una aproximación optimizada del cifrado de Vernam. La idea es construir una clave suficientemente larga a partir de la clave inicial corta utilizando un generador pseudoaleatorio. Este generador expande una clave corta (semilla) para obtener una mucho más larga. Además la operación de expansión ha de tener unas características determinadas porque la secuencia que se obtiene se utilizará para cifrar el texto claro.

Para cifrar un texto se suma cada bit del texto en claro con cada bit de la secuencias de cifrado*. El proceso de descifrado se realiza de manera igual, sumando bit a bit el texto cifrado con la secuencia binaria obtenida de la misma clave utilizando el mismo algoritmo determinístico.

Como la secuencia cifrante se obtiene a partir de un algoritmo determinístico, el cifrado en flujo ya no considera secuencias perfectamente aleatorias, que son las secuencias con la salida sino solamente **pseudoaleatorias**. Sin embargo, lo que se pierde en cuanto a seguridad, por no verificarse la condiciones de secreto perfecto, se gana en viabilidad práctica a la hora de utilizar este procedimiento de cifrado.

### 2.2 LFSR

Uno de los dispositivos más importantes para generar las secuencias pseudoaleatorias es **LFSR** - *registro de desplazamiento realimentado linealmente.*
Un **LFSR** es un dispositivo físico o lógico formado por n celdas de memoria y n puertas lógicas

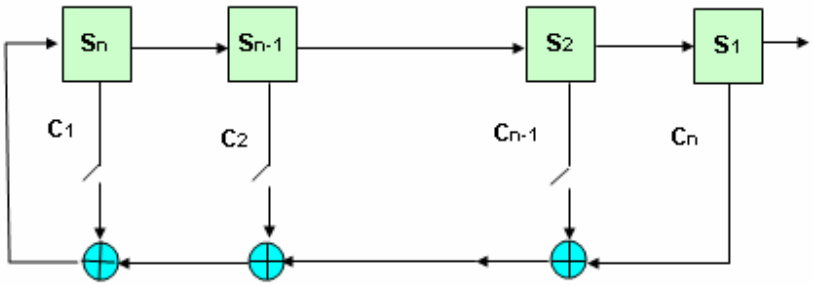


*Figura 3. Esquema del LFSR.*

Inicialmente las celdas contienen los valores de entrada (estado inicial de LFSR) y a cada impulso de reloj el contenido de la celda $s_i$ se desplaza a la celda $s_{i-1}$ realizando las operaciones asociadas. De esa manera se genera un nuevo elemento, $s_{n+1}$ , que se determina por

$$s_{n+1} = c_1 s_n + \ldots + c_n s_1$$

Hay que tener en cuenta que el estado inicial de LFSR no puede ser todos los ceros (el **estado absorbente**).

Todo registro de desplazamiento realimentado linealmente tiene asociado **un polinomio de conexión de grado n**

$$f(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_n x^n$$

donde $c_n \in \{0,1\}$ corresponde a los valores de las puertas lógicas.

Se puede determinar las características de **LFSR** de acuerdo con su polinomio de conexiones:

1. **Polinomio de conexiones factorizable:** los LFSRs que tienen como polinomios de conexiones polinomios factorizables generan las secuencias que dependen del estado inicial. El periodo de las secuencias generadas siempre es más corto que el periodo máximo que puede tener LFSR, que es $2^n$-1.

2. **Polinomio de conexiones irreductible:** las secuencias generadas por LFSRs con el polinomios de conexiones irreductibles no dependen del estado inicial, sino simplemente quedan desplazadas. El periodo de las secuencias generadas será un divisor de $2^n$-1.

3. **Polinomio de conexiones primitivo:** un LFSR con el polinomio de conexiones primitivo tiene la secuencia de salida de periodo máximo, $2^n$-1. Este periodo se obtiene para cualquier estado inicial, excepto el estado absorbente.

Los **LFSR** con el polinomio de conexiones primitivo cumplen con tres postulatos de Golomb [Anexo A], que determinan las propiedades de las secuencias pseudoaleatorias, gracias a la longitud máxima de su periodo y el hecho de pasar por todas los estados posibles.

Los **LFSR**s tienen una formulación matemática muy simple y además como se definen por medio de las celdas y las puertas lógicas se implementan muy fácilmente lo que permite obtener los generadores de gran velocidad. La rapidez de ejecución de cifrado en flujo es y será la mejor garantía de su vigencia. [2, p.56 ]

## 2.3 Generador de Geffe.

Sin embargo, el principal inconveniente de los generadores lineales para implementar los sistemas de cifrado de flujo es que son fácilmente predecibles. La predecibilidad de una secuencia periódica se denomina como complejidad lineal.

J.L.Massey ha definido la complejidad lineal de una secuencia como la longitud mínima (el nombre de celdas mínimo) de LFSR capaz de generarla. Por lo tanto una secuencia generada por un LFSR de longitud n tiene la complejidad lineal n, una complejidad muy baja comparándola con el periodo $2^n$-1.

Es evidente que para disminuir la predecibilidad de la secuencia de cifrado hay que aumentar la complejidad lineal de la secuencia de cifrado. Una manera de hacerlo es precisamente utilizar los generadores de secuencias basados en una combinación no lineal de varios registros de desplazamiento como en este caso el **generador de Geffe**.

Un generador Geffe está formado por **tres registros LFSR** con los siguientes parámetros:

- Grado de LFSR
- Polinomio de conexiones
- Estado inicial de registro

El esquema del generador Geffe es siguiente:

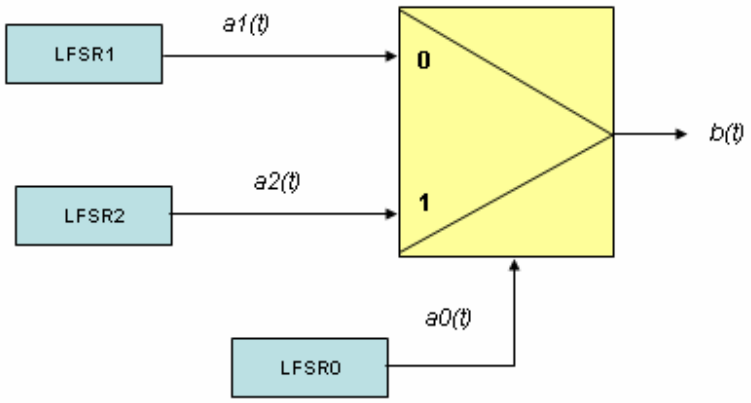


*Figura 4. Esquema de generador de Geffe.*

En forma algebraica normal la secuencia de salida puede expresarse como

$$b(t) = a_1(t) \oplus a_1a_0(t) \oplus a_2a_0(t)$$

Hay dos generadores LFSR1 y LFSR2 que generan las secuencias y un tercero LFSR0 que determina la función de salida.

A cada impulso de reloj se obtiene los valores de $LFSR^1$ y $LFSR^2$ y otro valor – de $LFSR^0$. Si valor de $LFSR^0$ es igual a 0, el valor de salida será valor de $LFSR^1$. En cambio, si el valor de $LFSR^0$ es igual a 1, el valor de salida será el valor de $LFSR^2$.

Con grado de polinomios de conexiones de los LFSR utilizados para la construcción de generador Geffe igual a n y en caso que los polinomios de conexiones sea primitivos la complejidad lineal de la secuencia de salida será

$$LC = n_1 + n_1n_0 + n_2n_0 \quad *$$

# Capítulo 3.

## IMPLEMENTACIÓN de GENERADOR de SECCUENCIA PSEUDOALEATORIA en INERCIPHER.

### 3.1 Implementación de generador de Geffe.

Para construir los tres LFSR a partir de la cadena de caracteres (la contraseña) se realizan los pasos siguientes.

- Se concatenan los 4 bits de menos peso de la codificación Unicode de cada uno de los caracteres de la contraseña.

- La cadena binaria obtenida se divide en tres bloques de g-1 bits y 3 bloques de g bits, donde g = grado LFSR.

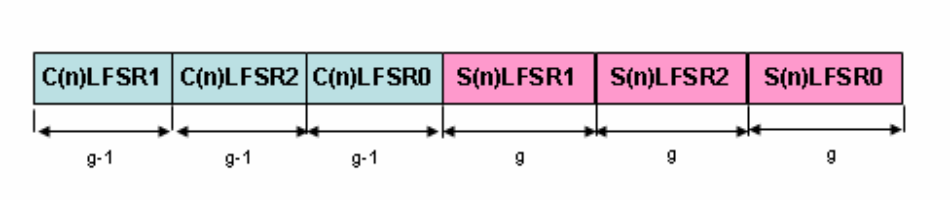


*Figura 5. Distribución de bits de la contraseña.*

- El primer bloque de g-1 bits define el polinomio de conexiones de $LFSR^1$ del generador Geffe. El coeficiente de mayor peso del polinomio se fija en 1 y el término independiente también. Los valores de la resta de los coeficientes se obtiene por el orden creciente, siendo C1 igual al primer bit de la cadena de entrada, el valor C2 igual al segundo bit etc.

- El segundo bloque de g-1 bits define el polinomio de conexiones de $LFSR^2$ del generador Geffe. Como en el caso de $LFSR^1$, el bit de mayor peso del polinomio de conexiones y el término independiente se fijan en 1 y el resto de los coeficiente se establecen por orden creciente.

- El tercer bloque de g bits define el polinomio de conexiones de $LFSR^0$ del generador Geffe ( el registro del control). El polinomio de conexiones se establece como en caso de $LFSR^1$ y $LFSR^2$.

- El cuarto bloque de g bits define el estado inicial de $LFSR^1$. El primer bit corresponde a la casilla $S_1$, el segundo a la $S_2$ etc..

- El quinto bloque de g bits define el estado inicial de $LFSR^2$.

- El sexto bloque de g bits define el estado inicial de LFSR[3].

El esquema del bloque-generador de las secuencias pseudoaleatorias es siguiente:

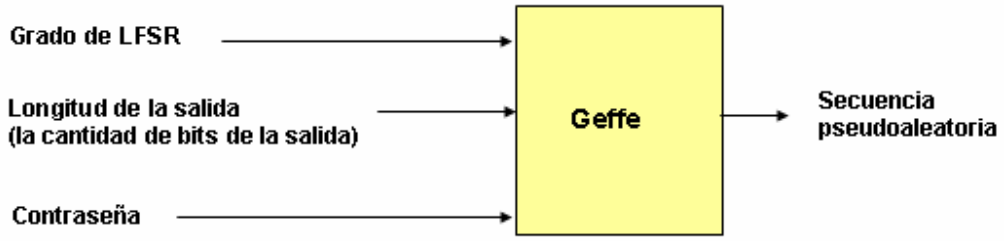


*Figura 6. Esquema general de bloque funcional Geffe.*

Para la aplicación INERCIPHER han sido utilizados los LFSR de grado 8 y se generan las secuencias pseudoaleatorias de 128 bits. La elección del grado de LFSR ha sido basada en garantía de seguridad por un lado, pero en mismo tiempo en la creación de una aplicación cómoda para usuario (el grado de LFSR elevado requiere la entrada de contraseñas demasiado largas).

### 3.2 Pruebas de funcionamiento de generador de Geffe.

Para realizar las pruebas de funcionamiento del bloque-generador Grado de LFSR y Longitud de salida se entran como los argumentos y la contraseña se pide al usuario en momento de ejecución de programa. Para realizar las pruebas ha sido generado el proyecto *java Geffe* con el código que contiene las salidas de los resultados intermedios por la pantalla (ver los vectores de teste) de ese modo se ha conseguido comprobar que cada uno de LFSR ha sido implementado correctamente y el funcionamiento del generador en general es correcto.

Los vectores de teste* utilizados para comprobar si la implementación de los LFSRs y del generador Geffe es correcta son siguientes:

| |
|---|
| Grado de LFSR: 3 Bytes de salida : 4<br>Contraseña : “geffe” |
| Bits obtenidos de contraseña : 01110101011001100101 |
| Salida LFSR1 : 01011100101110010111001011100101 |
| Salida LFSR2 : 11001100110011001100110011001100 |
| Salida LFSR0 : 01110010111001011100101110010111 |
| Salida Geffe : 4cdcf8e4 |

| |
|---|
| Grado LFSR : 8 Bytes de salida : 10<br>Contraseña : “criptografia” |
| Bits obtenidos de la contraseña:<br>0011001010010000010011110111001000010110100100 01 |
| Salida LFSR1 : 1110111000010001000000110101101100111001<br>1110100110010001100110000010111101101010 |
| Salida LFSR2 : 0100001011101110000110001110011001000101<br>0010010101011001111011011111110101000110 |
| Salida LFSR0 : 1101001001010011110010001101010000111111<br>1011101101111010001011001011111000100000 |
| Salida Geffe : 6e420bcf0561d9bcbd4a |

Además de las excepciones propias del lenguaje Java se tratan como excepciones las situaciones siguientes:

- El estado inicial de LFSR es el estado absorbente;
- La longitud de la contraseña no permite generar los LFSR del grado deseado.

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\User>cd \Geffe

C:\Geffe>java Geffe
Los argumentos de entrada han de ser numeros enteros positivos

C:\Geffe>java Geffe 3
Los argumentos de entrada han de ser numeros enteros positivos

C:\Geffe>java Geffe 3 4
Entre su contrasena:
*****
01110101011001100101
01011100101110010111001011100101
11001100110011001100110011001100
01110010111001011100101110010111

4cdcf8e4

C:\Geffe>java Geffe 8 10
Entre su contrasena:
**********
0011001010010000010011110111001000010110100100 01
11101110000100010000001101011011001110011110100110010001100110000010111101101010
01000010111011100001100011100110010001010010010101011001111011011111110101000110
11010010010100111100100011010100001111111011101101111010001011001011111000100000

6e420bcf0561d9bcbd4a

C:\Geffe>java Geffe 8 10
Entre su contrasena:
*****
01110101011001100101

Error en la creacion de los polinomios de conexion.
La contrasena para este grado ha de ser
 mas larga

C:\Geffe>java Geffe 3 4
Entre su contrasena:
**********
0011001010010000010011110111001000010110100100 01

Estado inicial de LFSR0 es el estadoabsorbente. Cambie la contraseña
 o el grado de LFSR.

C:\Geffe>
```

*Figura 7. Captura de la pantalla al realizar las pruebas de Generador de Geffe.*

* Los vectores de teste han sido proporcionados por los profesores de Criptografía de la UOC.

# Capítulo 4.

## BLOQUE CIFRADOR/DESCIFRADOR

### 4.1 Cifrado de bloque

Una alternativa al cifrado de flujo es el **cifrado de bloque**. Este tipo de cifrado se incluye en grupo de criptosistemas de clave compartida porque la clave que se utiliza para cifrar y descifrar es la misma. La diferencia básica entre el cifrado de flujo y cifrado de bloque es la utilización de la memoria en los algoritmos de cifrado.

El cifrado de flujo utiliza una clave diferente para cada bit de información. Esta clave depende no solamente del estado inicial de generador, sino también del estado de generador en el momento de cifrar un bit concreto, por lo cula dos bits iguales podrán cifrarse de maneras diferentes en función de estado en que se encuentra el generador. En cambio, los cifrados de bloque actúan sin memoria y el texto cifrado solo puede depender del texto claro y de la clave: en el cifrado de bloque dos textos iguales se cifran de la misma manera cuando se utiliza la misma clave.

Los cifrados de bloque cifran el texto original agrupando los símbolos en grupos (bloques) de dos o más elementos y se componen de 4 procesos:

1. Transformación inicial.
2. Una función criptográfica débil iterada r veces o “vueltas”.
3. Transformación final.
4. Algoritmo de expansión de clave.

La transformación inicial puede tener una o dos funciones: la primera consiste en aleatorizar simplemente los datos de entrada (para ocultar bloques de datos de todo ceros o unos, etc.), careciendo de significación criptográfica si no depende de la clave. La segunda función solamente tiene lugar en algunos criptosistemas determinados, como por ejemplo IDEA y RC5, tiene significado criptográfico , dificultando ataques por criptoanálisis lineal o diferencial; en este caso es función de la clave.

Las vueltas intermedias consisten en una función no lineal complicada de los datos y la clave, que puede ser unidireccional o no. La función no lineal puede estar formada por una sola operación muy compleja o por la sucesión de varias transformaciones simples.

Las vueltas intermedias se enlazan por sumas módulo 2 bit a bit con los datos que vienen de la transformación inicial o de las vueltas precedentes; de esta manera se posibilita que se produzca una “involución” cuando se repite el proceso el proceso de forma idéntica (pero eligiendo las claves de descriptación en orden inverso), obteniéndose de esta forma los datos de partida.

Las vueltas intermedias no han de formar grupo, para que el conjunto de varias pasadas sucesivas con sus subclaves correspondientes no sea equivalentes a una pasada única con una subclave diferente.

La transformación final sirve para que las operaciones de encriptación y desencriptación sean simétricas. Cuando las vueltas de encriptación son de una sola operación, separadas por suma módulo 2bit a bit, esta transformación se limita a realizar la

operación inversa de la transformación inicial. Sin embargo, en los sistemas donde las vueltas de encriptación acaban con una operación que afecta a todos los bits de bloque, la transformación de salida debe realizar tanto la función inversa de esta operación como la inversa de la transformación inicial.

El algoritmo de expansión de clave tiene por objeto convertir la clave de usuario, normalmente de la longitud limitada entre 32 y 256 bits, a un conjunto de subclaves que pueden estar constituidas por varios cientos de bits en total. Conviene que sea unidireccional y que el conocimiento de una o varias subclaves intermedias no permita deducir las subclaves anteriores o siguientes. Además se ha de cuidar que las subclaves producidas no constituyan un pequeño subconjunto monótono de todas las posibilidades [2, p.61]

## 4.2 AES

Para la implementación de la aplicación INERCIPHER ha sido elegido el cifrado de bloque **AES** como el más seguro de los cifrados de bloque en la actualidad.

En 1996 el Instituto Nacional de Estándares y Tecnología (Nacional Institute of Standard and Tecnology, NIST) dio los primeros pasos para la creación de un Estándar de Cifrado Avanzado (Advanced Encryption Estándar, AES). Su objetivo fue desarollar una especificación para encontrar un algoritmo de cifrado que sustituyera al anticuado DES, de manera que el nuevo algoritmo fuese capaz de proteger a información sensible de los ciudadanos y del gobierno hasta bien entrado el siglo XXI. El algoritmo seleccionado sería utilizado por gobierno de Estados Unidos para proteger la información sensible no clasificada y voluntariamente por el sector privado.

En septiembre de 1997 se realizó una convocatoria para la presentación de algoritmos; los propuestos tenían que soportar obligatoriamente la longitud de bloques de 128 bits y una longitud de clave 128, 192 y 256 bits, al margen de cualesquiera otras longitudes posibles. La razón por la que se ha exigido la posibilidad de utilizar claves de 128, 192 y 256 bits es que, a pesar de que un clave de 128 bits hoy día es totalmente imposible de romper con un ataque de fuerza bruta, no puede asegurarse que en un futuro próximo los avances en la ciencia de computación no vayan a permitir reventar claves de esa longitud.

A la citada convocatoria acudieron diversos diseñadores. La selección se levó a cabo en varias etapas eliminatorias, y se izo de forma pública., basándose en las críticas, evaluación y criptoanálisis realizados por la comunidad científica internacional. Finalmente, el 2 de octubre de 2000, el NIST anunció el algoritmo ganador: el Rijndael, propuesto por los belgas Vincent Rijmen y Joan Daemen.

Los criterios considerados en la selección fueron los siguientes:

1. Seguridad (el esfuerzo necesario para criptoanalizar los algoritmos).
2. Eficiencia computacional.
3. Requisitos de memoria.
4. Adecuación hardware y software.
5. Simplicidad del diseño.
6. Flexibilidad.
7. Requisitos de licencia.

El 26 de noviembre de 2001 el NIST publicó la norma definitiva, que entro en vigor el 26 de mayo de 2002.[ 8, FIPS Pubs].

El Rijndael es un cifrador que opera con bloques de longitud igual a Nb palabras de 32 bits; los valores de Nb pueden ser 4, 6 y 8; la longitud de clave es de Nk palabras de 32 bits; los valores de Nk pueden ser 4,6 y 8. Es decir, que las longitudes de clave y bloque pueden ser de 128, 192 y 256 bits.

El Rijndael es fácilmente adaptable a cualquier longitud de bloque y/o clave múltiplo de 32 bits. La versión finalmente aprobada por NIST restringe la longitud de bloque a 128 bits (Nb = 4), mientras que la clave puede ser de 128, 192 o 256 bits.

El AES es un cifrador iterado, relativamente sencillo, que emplea funciones invertibles y opera con bloques enteros. Al resultado obtenido en cada paso del algoritmo se le denomina el Estado. Es un conjunto de tantos bits como la longitud del bloque. Los bits adyacentes se agrupan de ocho en ocho, formando bytes, y estos en una tabla cuadrada de cuatro filas y 4 columnas (bloque de 128 bits).

### 4.3 Esquema de AES.

Esquema general del AES se presenta en la figura siguiente (los diferentes elementos de un cifrado de bloque, explicados en la p. están presentes en colores diferentes):

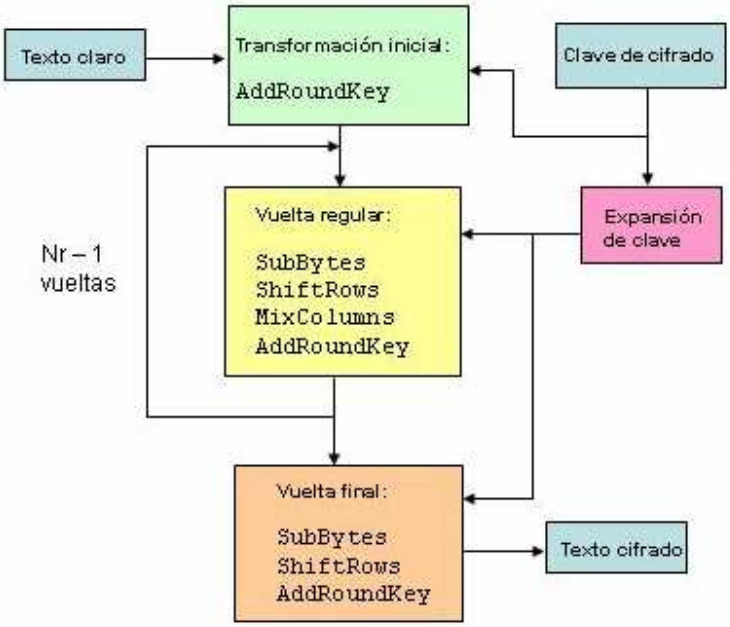


*Figura 8. Esquema general de AES.*

La transformación que tiene lugar en cada vuelta regular de cifrado está compuesta por cuatro transformaciones diferentes:

1. `SubBytes()` : sustitución no lineal de los bytes.
   Los autores del AES dan una fórmula matemática para su construcción orientada a cumplir con los siguientes criterios:
   - Minimizar la correlación entre la entrada y salida.
   - Minimizar la probabilidad de propagación de las diferencias.
   - Maximizar la complejidad de la expresión algebraica de la transformación.

2. `ShiftRows()` : desplazamiento de las filas de Estado cíclicamente, con diferentes saltos.
   Se permutan cíclicamente los contenidos de las filas del Estado. Esta transformación contribuye a lograr la difusión, siendo resistente a los ataques diferenciales.

3. `MixColumns()` : mezcla de columnas.
   Esta transformación opera sobre el Estado, columna por columna. Consiste en multiplicar cada columna por una matriz de la manera siguiente:

4. `AddRoundKey()` : suma módulo 2 (XOR) con la subclave correspondiente.
   En esta transformación se suman módulo 2 (XOR) los bits del Estado y los bits de la subclave de vuelta Kr correspondiente.

La vuelta final es casi igual a las vueltas regulares. La diferencia consiste en la no existencia de la tercera transformación.

El descifrado se realiza efectuando las operaciones inversas de las empleadas en cifrado, en orden inverso al utilizado en éste.

### 4.4 Esquema de clave en AES.

La longitud de la clave en AES puede ser 128, 192 o 256 bits y es evidente que es necesario generar muchas subclaves a partir de ella: la subclave inicial, más Nr subclaves de vuelta, cada una de ellas de cuatro palabras de 32 bits. Las subclaves requeridas se generan por derivación de la clave de cifrado mediante un procedimiento de expansión, cuyo esquema varia en función de la longitud de la clave de cifrado. En la figura se presenta el esquema para Nk = 4 (clave de 128 bits), la clave ha sido utilizada en el bloque cifrador/descifrador de INERCIPHER.

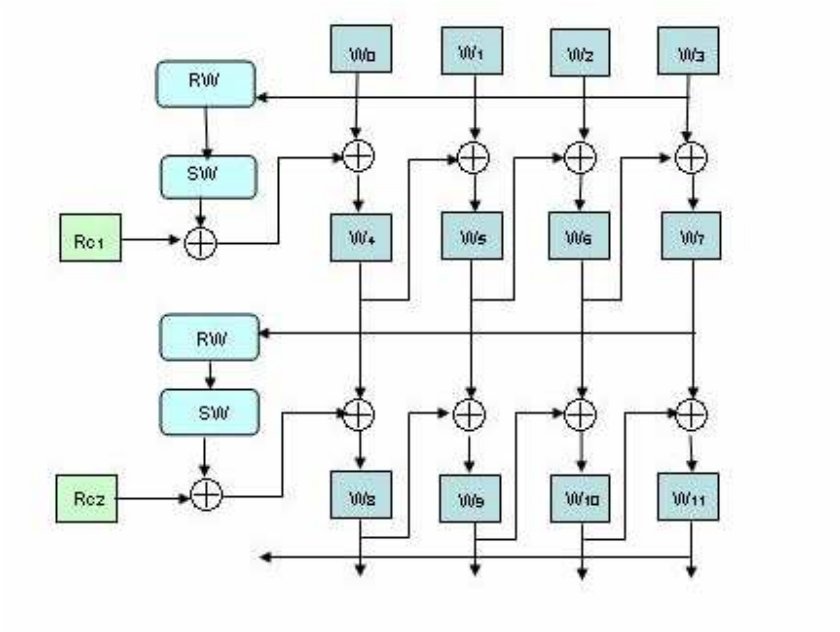


*Figura 9. Esquema de expansión de clave en AES.*

La función `SubWord()` (SW en el esquema) consiste en aplicar la transformación `SubBytes()` a cada uno de los cuatro bytes de cada palabra de entrada.

La función `RotWord()` (RW en el esquema) consiste en rotar cíclicamente a izquierda byte a byte cada palabra de entrada. Por ejemplo la palabra [$b_0$, $b_1$, $b_2$, $b_3$] se transformaría en palabra [$b_1$, $b_2$, $b_3$, $b_0$].

`Rcon[n]` ($R_{Ci}$ en el esquema) es una constante de 32 bits dependiente del número de vuelta. Los valores posibles se presentan en la tabla siguiente:

| n | Rcon[n] | n | Rcon[n] |
|---|---|---|---|
| 1 | 01000000 | 6 | 20000000 |
| 2 | 02000000 | 7 | 40000000 |
| 3 | 04000000 | 8 | 80000000 |
| 4 | 08000000 | 9 | 1b000000 |
| 5 | 10000000 | 10 | 36000000 |

*Tabla 1. Valores de la constante $R_{Cn}$*

### 4.5 Modo de implementación de AES en bloque Cifrador/Descifrador.

El cifrado se puede realizar de varios modos, según la configuración que se adopte. Para cualquier cifrado en bloque se puede utilizar los modos siguientes:

- ECB (Electronic Codebook – Libro Electrónico de Códigos)

- OFB (Output Feedback, realimentación de salida)
- CFB (Cipher Feedback, realimentación del texto cifrado)
- CBC (Cipher Block Chaining, encadenamiento de bloques cifrados)

Para la implementación del bloque Cifrador/Descifrador ha sido utilizado el modo CBC porque este método es preferido para los cifrados en bloque cuando tasa de errores es baja [2, p.134].

El modo CBC consiste en el encadenamiento de los bloques para el cifrado de manera que se crea una dependencia de cifrado de cada bloque con el inmediato anterior. Para cifrar el primer bloque se necesita un bloque inicial aleatorio $C_0$ (vector inicial) que no hace falta que sea secreto. El resultado del cifrado – el bloque cifrado – se utilizará como el vector para el cifrado del bloque de texto siguiente:

$$C_i = E_k ( M_i \oplus C_{i-1} )$$

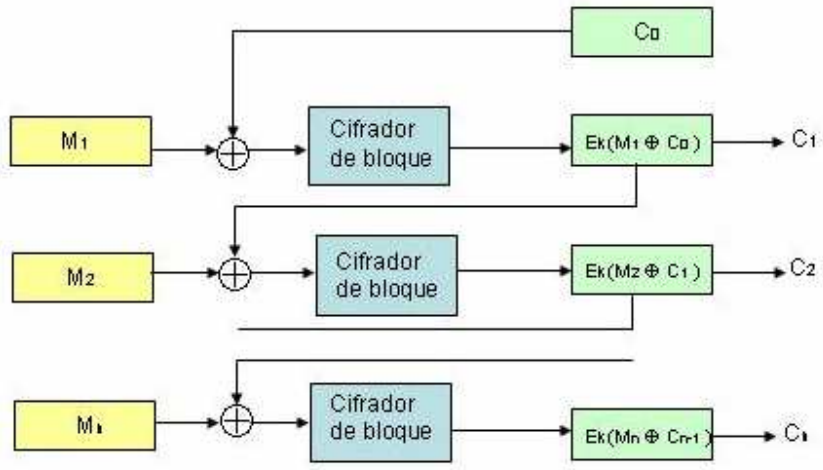


*Figura 10. Esquema de modo de cifrado CBC.*

Las propiedades de este modo de cifrado son siguientes:

1. Asegura la confidencialidad.
2. Mala recuperación de sincronismo frente a alteraciones graves de transmisión.
3. Moderada insensibilidad frente a ruido y errores.
4. Detección de trozos del criptograma alterados.
5. Iguales mensajes con igual clave e igual VI(vector inicial) producen en diferentes momentos iguales criptosistemas. Si el VI es diferente, los criptosistemas serán diferentes.

### 4.6 Implementación de bloque Cifrador/Descifrador.

La implementación de bloque CIFRADOR/DESCIFRADOR en la aplicación INERCIPHER se realiza utilizando las clases y métodos proporcionados por la librería de Java.

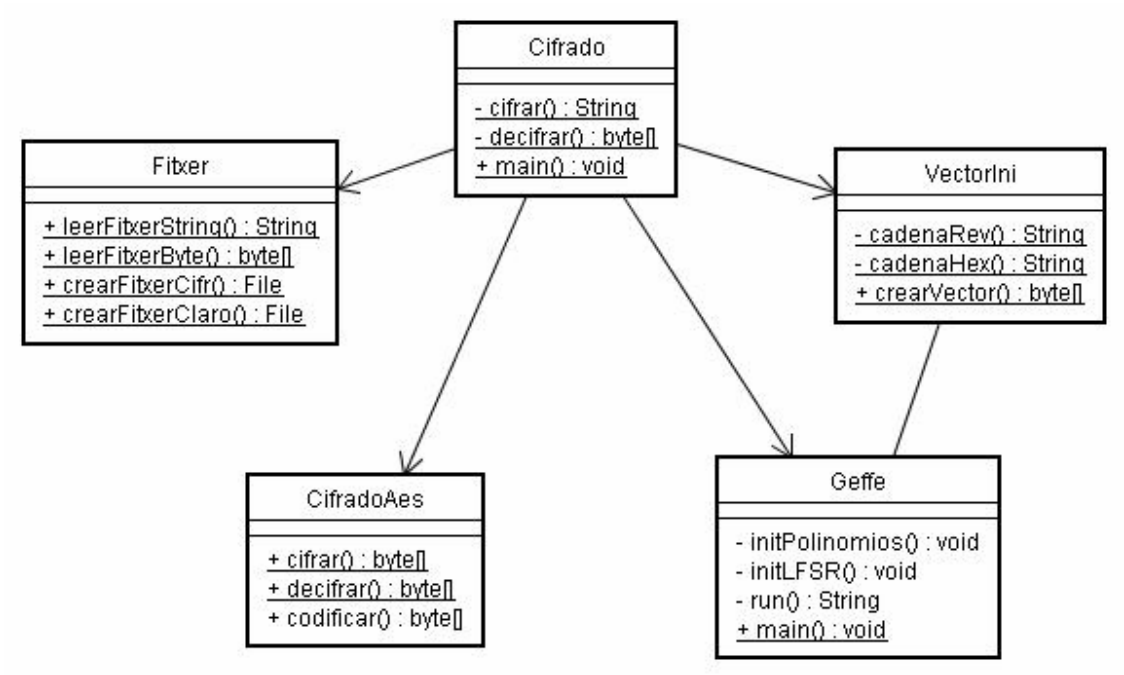


*Figura 11. Diagrama de clases del bloque Cifrador/Descifrado**

La clase principal del bloque (la clase que implementa el método main()) es Cifrado.java. Los argumentos del método `static void main(String[] args)` de la clase Cifrado.java son la secuencia pseudoaleatoria de cifrado obtenida a partir de la contraseña del usuario (bloque Generador de Geffe), vector inicial que se obtiene de la cadena de bits de la contraseña** , nombre de fichero y el código de operación solicitada (cifrado o descifrado). El fichero para cifrar/descifrar y la operación en concreto se eligen utilizando las estructuras de la interfaz gráfica en modo interactivo.

La aplicación permite cifrar/descifrar los archivos uno por uno (la clase FileDialog de la librería de Java permite realizar la búsqueda de un archivo determinado en los dispositivos de almacenamiento).

*En el diagrama de clases solo están incluidos los clases principales para el proceso de de cifrado/descifrado con los métodos más importantes para realizar estas funciones para presentar el esquema general del bloque.

** El proceso de creación de vector inicial para el cifrado está escrito detalladamente en el Anexo B.

El archivo cifrado con la extensión **.cifr** se guarda en el mismo directorio donde está el archivo con texto claro.

Después de realizar la operación ***Cifrar*** se recomienda eliminar el archivo claro (la interfaz gráfica dispone de esta opción). En caso de eliminar el archivo con texto claro (la opción recomendada) el usuario ha de tener en cuenta que no se podrá recuperar los datos guardados en este archivo si no dispone de la contraseña de usuario correcta (la contraseña que ha sido utilizada para realizar el cifrado). Además la aplicación no dispone de un sistema de almacenamiento de contraseñas, lo que significa que en caso de olvidar la contraseña el usuario no podrá descifrar los archivos. Teniendo en cuenta estos aspectos se recomienda hacer las copias de los archivos con texto claro en alguno de los dispositivos de memoria extraíbles y guardarlos en un sitio seguro.

La aplicación no dispone de la opción de cifrado de las carpetas. En caso que un usuario quisiera realizar el cifrado de cada uno de los archivos de una carpeta pero sin hacerlo uno por uno, puede generar previamente un archivo comprimido tipo .zip y cifrar este archivo.
La principal ventaja de este método es la posibilidad de cifrar varios archivos ejecutando la opción cifrado solamente una vez, pero el tiempo de realización de cifrado será bastante importante y además todos los archivos estarán cifrados como un único archivo lo que hace imposible ver la estructura de directorio y tampoco sería posible descifrar un archivo en concreto (para recuperar un archivo hay que descifrar archivo .zip.cifr que dependiendo del tamaño de archivo puede ser un proceso bastante lento).

Los archivos de prueba de funcionamiento se presentan agrupados en el archivo *iermolova_archivos_de_prueba.zip*. Los archivos claros con sus códigos MAC correspondientes están agrupados en *archivos_claros.zip*. El archivo *archivos_claros_desc.zip* contiene además de los archivos cifrados, el resultado de descifrado con la contraseña correcta. Los archivos con texto claro agrupados en *archivos_claros_error.zip* han sido obtenidos realizando descifrado con la contraseña errónea.

Las pruebas como en caso de Cifrador/Descifrador han sido realizadas utilizando contraseña "*uocinercipher*".

# Capítulo 5.

## Bloque DocIntegrity.

### 5.1 Autenticación de mensajes (MAC).

Inicialmente la aplicación INERCIPHER disponía de una sola funcionalidad cifrado/descifrado de los archivos. Sin embargo, pueden existir los documentos guardados en los dispositivos de memoria que disponen de permiso de lectura para los usuarios permitidos pero en mismo tiempo son críticos desde punto de vista de la autenticidad y la integridad. Por esta razón la funcionalidad de la aplicación ha sido complementada por la opción que permite generar el código de autentificación MAC (Message Authentification Code) de los documentos y verificar posterior mente la integridad de los documentos.

La autenticación de mensajes se realiza mediante una sencilla manipulación criptográfica: se genera el código MAC (su versión cifrada condensada) de un documento y se guarda o en mismo dispositivo de memoria o en caso de extrema seguridad – en un dispositivo de memoria extraíble que se guardará posteriormente en un sitio seguro. Antes de comenzar el trabajo con el documento crítico se comprueba su integridad: se vuelve a generar el código MAC del documento utilizando la misma contraseña (se realiza el cifrado con la misma clave) y este código se compara con el código MAC del documento original. En caso que sean iguales ambos códigos, se puede concluir que el documento es íntegro.

### 5.2 Algoritmo AesXCbcMAC.

Para generar el código MAC de los documentos ha sido implementado el algoritmo AesXCbcMac, utilizado por ejemplo como medio de autentificación en algunos mecanismos del protocolo de red IPSec. Este algoritmo utiliza el cifrador de bloque AES con una longitud de clave 128 bits.

Para implementar el algoritmo AesXCbcMac han sido realizados los pasos siguientes:

1. El documento inicial M se divide en n bloques. Los primeros n-1 bloques tienen una dimensión de 128 bits, el último bloque – la dimensión entre 1 y 128 bits.
2. A partir de una clave K de 128 bitas, se derivan 3 subclaves K1, K2 y K3 de manera siguiente:

   a. $K1 = E_K(\ 0x01010101010101010101010101010101\ )$

   b. $K2 = E_K(\ 0x02020202020202020202020202020202\ )$

   c. $K3 = E_K(\ 0x03030303030303030303030303030303\ )$

   donde Ek simboliza la operación de cifrado AES con la clave K.

3. Se define E[0] = 0x00000000000000000000000000000000
4. Para cada bloque M[i] con i = 1 …n-1 se realiza $E[i] = E_{K1}(\ M[i]\ XOR\ E[i-1]\ )$.

5. Para último bloque M[n] :
   si la dimensión de M[n] es 128 bits se realiza $E[n] = E_{K1}( M[n] \text{ XOR } E[n-1] \text{ XOR } K2 )$, sino se concatena un único bit "1" en el bloque M[n] seguido de un número de bits "0" (es posible que no hace falta ninguno) necesarios para conseguir que la dimensión del bloque M[n] sea de 128 bits. Y se realiza la operación siguiente:

$$E[n] = E_{K1}( M[n] \text{ XOR } E[n-1] \text{ XOR } K3 )$$

El valor de **E[n]** es el código de autenticación MAC[17].

### 5.3 Implementación del bloque DocIntegrity.

Las clases principales que constituye bloque DocIntegrity son siguientes:

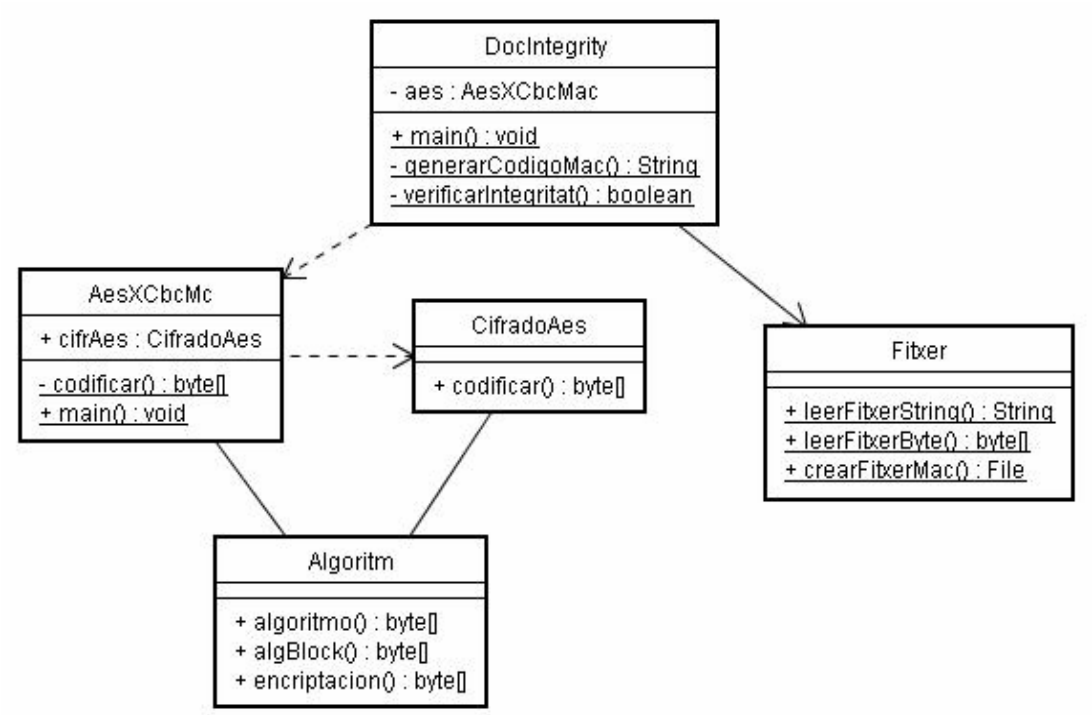


*Figura 12. Diagrama de clases de bloque DocIntegrity.*

La clase principal del bloque es DocIntegrity, que incluye el método de verificación de integridad. La implementación de algoritmo se realiza principalmente en la clase Algoritm. Como en caso de bloque Cifrador/Descifrador el cifrado AES se realiza con los métodos implementados en clase CifradoAes (esta clase se utiliza por ambos bloques funcionales, igual como clase Fitxer) utilizando la librería estándar de Java.

### 5.4 Pruebas de algoritmo AesXCbcMac.

Para comprobar si la implementación del algoritmo AesXCbcMac es correcta han sido utilizados los vectores de teste siguientes [17]:

```
These test cases were provided by John Black, co-author of the
   XCBC-MAC algorithm, who verified them with 2 independent
   implementations.  All values are hexadecimal numbers.

   Test Case #1   : AES-XCBC-MAC-96 with 0-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : <empty string>
   AES-XCBC-MAC   : 75f0251d528ac01c4573dfd584d79f29
   AES-XCBC-MAC-96: 75f0251d528ac01c4573dfd5

   Test Case #2   : AES-XCBC-MAC-96 with 3-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 000102
   AES-XCBC-MAC   : 5b376580ae2f19afe7219ceef172756
   AES-XCBC-MAC-96: 5b376580ae2f19afe7219cee

   Test Case #3   : AES-XCBC-MAC-96 with 16-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 000102030405060708090a0b0c0d0e0f
   AES-XCBC-MAC   : d2a246fa349b68a79998a4394ff7a263
   AES-XCBC-MAC-96: d2a246fa349b68a79998a439

   Test Case #4   : AES-XCBC-MAC-96 with 20-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 000102030405060708090a0b0c0d0e0f10111213
   AES-XCBC-MAC   : 47f51b4564966215b8985c63055ed308
   AES-XCBC-MAC-96: 47f51b4564966215b8985c63

   Test Case #5   : AES-XCBC-MAC-96 with 32-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 000102030405060708090a0b0c0d0e0f10111213141516171819
                    1a1b1c1d1e1f
   AES-XCBC-MAC   : f54f0ec8d2b9f3d36807734bd5283fd4

   AES-XCBC-MAC-96: f54f0ec8d2b9f3d36807734b

   Test Case #6   : AES-XCBC-MAC-96 with 34-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 000102030405060708090a0b0c0d0e0f10111213141516171819
                    1a1b1c1d1e1f2021
   AES-XCBC-MAC   : becbb3bccdb518a30677d5481fb6b4d8
   AES-XCBC-MAC-96: becbb3bccdb518a30677d548

   Test Case #7   : AES-XCBC-MAC-96 with 1000-byte input
   Key (K)        : 000102030405060708090a0b0c0d0e0f
   Message (M)    : 00000000000000000000 … 00000000000000000000
                    [1000 bytes]
```

Las pruebas han sido realizadas utilizando la contraseña "*ingaermolovaermolova*".

Los resultados de las pruebas realizadas se presentan en la figura siguiente:

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\User\Escritorio>cd cifr

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 0
AES-XCBC-MAC  : 75f0251d528ac01c4573dfd584d79f29

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0
02

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 000102
AES-XCBC-MAC  : 5b376580ae2f19afe7219ceef172756f

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0
0203040506070809 0a0b0c0d0e0f

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 000102030405060708090a0b0c0d0e0f
AES-XCBC-MAC  : d2a246fa349b68a79998a4394ff7a263

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0
0203040506070809 0a0b0c0d0e0f10111213

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 000102030405060708090a0b0c0d0e0f10111213
AES-XCBC-MAC  : 47f51b4564966215b8985c63055ed308

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0
0203040506070809 0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
AES-XCBC-MAC  : f54f0ec8d2b9f3d36807734bd5283fd4

C:\Documents and Settings\User\Escritorio\cifr>java AesXCbcMac 000102030405060708090a0b0c0d0e0f 0
0203040506070809 0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f2021

Key (K) : 000102030405060708090a0b0c0d0e0f
Missatge (M)  : 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f2021
AES-XCBC-MAC  : becbb3bccdb518a30677d5481fb6b4d8

C:\Documents and Settings\User\Escritorio\cifr>
```

*Figura 13. Captura de la pantalla al realizar las pruebas con vectores de teste.*

Para realizar las pruebas de implementación del algoritmo AesXCbcMac ha sido ejecutado: `java AesXCbcMac [key] [message_byte]`* desde la línea de comandos.

Los códigos MAC de los archivos con texto claro, generado utilizando contraseña "*uocinercipher*", están incluidos en el archivo *archivos_claros.zip*.

Los archivos de prueba de funcionamiento del verificador de la integridad se presentan agrupados en el archivo *archivos_prueba_MAC.zip,* que también está incluido en el archivo general de las pruebas *iermolova_archivos_de_prueba.zip*. El fichero *archivos_prueba_MAC.zip* MAC contiene los archivos **.mac** con indicador **m** (*nom_ficher**m**.mac*) donde se guarda el mismo código MAC, que está en los archivos *nom_ficher.mac* de *archivos_claros.*, en cambio los archivos *nom_ficher**m*** tienen algunas modificaciones respecto a los archivos *nom_ficher* **.**

*La clase AesXCbcMac implementa el método main() lo que ha facilitado la realización de pruebas.

# Capítulo 6.

## Interfaz gráfica.

Para facilitar el uso de la aplicación ha sido creada la interfaz gráfica que permite al usuario el trabajo interactivo. Además los mensaje informativos o de error guian al usuario paso a paso lo que facilita el uso de la interfaz.

La ejecución de la aplicación INERCIPHER con interfaz gráfica presenta en la figura siguiente:

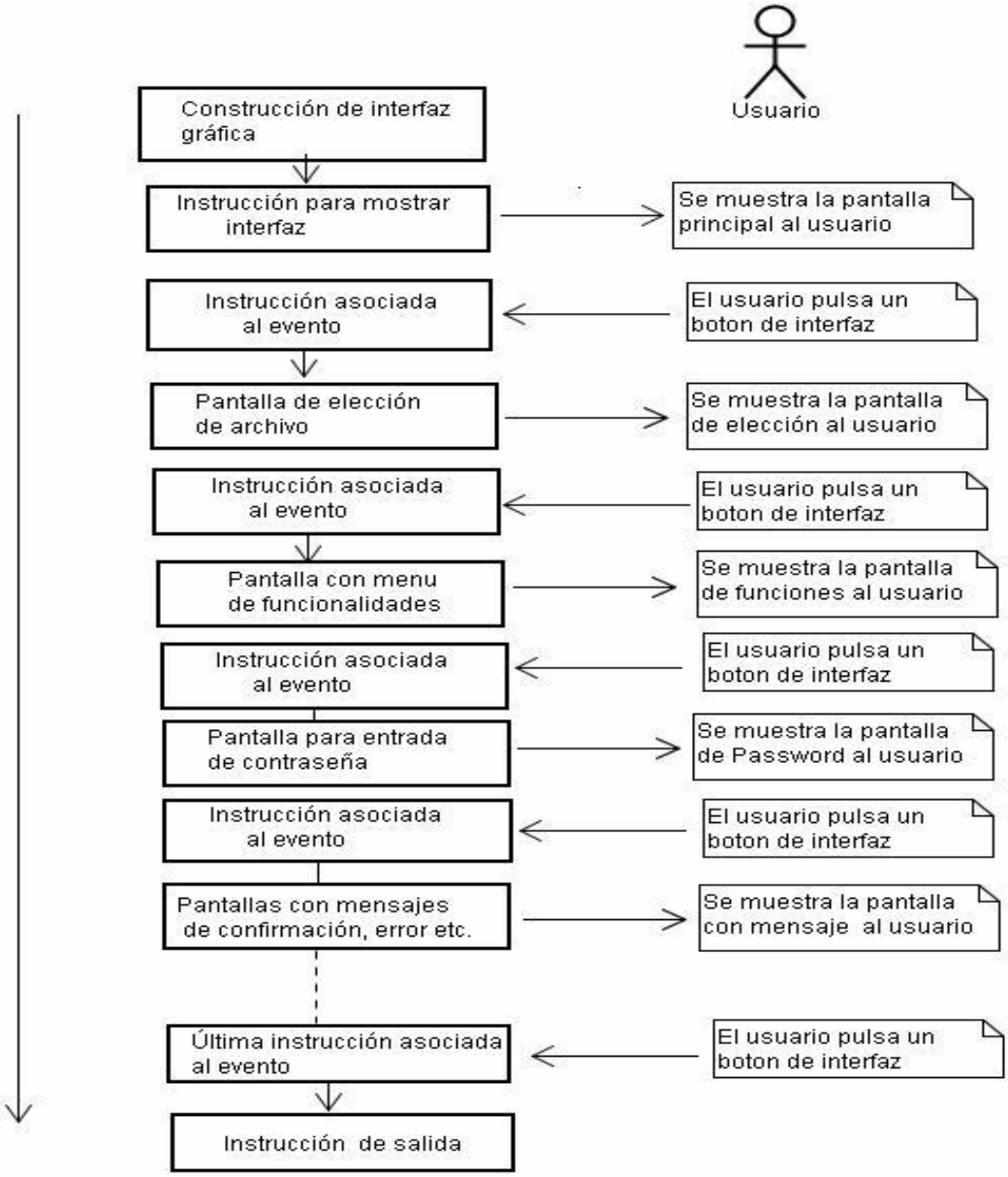


*Figura 14. Ejecución de INERCIPHER con interfaz gráfica.*

La explicación detallada del funcionamiento de la interfaz se presenta en el Manual de usuario (Anexo C). Además se adjunta una copia de manual con los archivos de aplicación.

Para hacer el uso de la aplicación más cómodo las respuestas a las preguntas más frecuentes sobre el funcionamiento de la aplicación están incluidas en Menu principal (opción Ayuda).

# Capítulo 7.

## Conclusiones.

La aplicación INERCIPHER es una herramienta para realizar el cifrado/descifrado de los archivos y/o la verificación de la integridad de documentos sencilla, pero en mismo tiempo segura.

La seguridad de los cifrados está basada en la utilización de los algoritmos modernos, como AES o AesXCbcMac y en la implementación del generador de Geffe, que además del proceso de construcción de LFSR a partir de los caracteres de contraseña complicado, utiliza el grado de los generadores lineales bastante importante. Por otro lado la utilización de generador de Geffe para generación de la secuencia pseudoaleatoria garantiza muy buena velocidad de ejecución de aplicación en general.

Por razones de seguridad no ha sido implementado ningún sistema para almacenar contraseñas, ni para poder recuperar el texto en claro de documentos cifrados en caso de olvidar la contraseña de cifrado. Desde punto de vista de usuario este aspecto puede valorarse como un inconveniente, aunque está comentado explícitamente en el Manual de usuario con las advertencias correspondientes.

Como la aplicación no necesita la instalación, no hay peligro que la utilización de INERCIPHER pueda provocar los errores al nivel de SO de equipo, ni otros programas instalados. Además la aplicación no ocupa la cantidad de memoria importante y puede ser trasladada fácilmente a otros equipos.

# Glosario.

**AES:** *Advanced Encryption Standard.* Criptosistema Rijndael, que cifra bloques de 128 bits por medio de una clave que puede variar de longitud entre 128, 192 o 256 bits.

**AesXCbcMac:** algoritmo que se utiliza para generar el código MAC de un mensaje. Ha sido publicado en septiembre de 2003.

**CBC:** *Cipher Block Chaining.* Modo de cifrado de bloque en cual se crea encadenamiento de bloques de manera que el cifrado de un bloque depende del anterior. El bloque inicial para el cifrado es aleatorio.

**Complejidad lineal de una secuencia:** número de celdas de LFSR más corto que es capaz de generar una secuencia.

**Criptosistema de clave compartida:** criptosistema en que el emisor y el receptor comparten una sola clave que se hace servir para cifrar y para descifrar.

**Cifrado de flujo**: sistema de cifrado que utiliza un generador pseudoaleatoriopara cifrar un mensaje, sumando bit a bit el texto en claro con la secuencia pseudoaleatoria que se obtiene con el generador.

**Generador de Geffe:** es un generador no lineal de secuencias pseudoaleatorias formado por tres registros LFSR con los siguientes parámetros: grado de LFSR, polinomio de conexiones y estado inicial de registro.

**Generador lineal:** generador de secuencias de bits que solamente ejecuta operaciones lineales sobre los elementos de entrada para obtener la secuencia de salida.

**ECB:** *Electronic Code Book.* Modo de cifrado de bloque en cual el cifrado de bloques es independiente uno de otro y se realiza con la misma clave.

**Estado de un LFSR:** conjunto de valores conteidos en cada celda de un LFSR en un instante de tiempo.

**Estado absorbente de LFSR:** es el estado inicial de LFSR , cuando todas las celdas tienen valor cero.

**Función hash:** función que da como la salida un resumen de longitud fija a partir de una entrada consistente en un mensaje arbitrariamente largo.

**LFSR:** es un dispositivo físico o lógico formado por n celdas de memoria y n puertas lógicas para generación de las secuencias pseudoaleatorias.

**MAC:** *Message Authentification Code.* Versión cifrada y condensada de un texto claro.

**Polinomio de conexión de un LFSR:** polinomio que determina o que es determinado por la función lineal de realimentación de LFSR.

# Bibliografía.


1. J. Cariacedo Gallardo. *Seguridad en redes telemáticas.* McGRAW-HILL, 2004
2. A. Fúster Sabater, D. Martínez, L. Hernández Encinas, F. Montoya Vitini, J. Muñoz Masqué. *Técnicas criptográficas de protección de datos.* Ra –MA, 2004.
3. B. Ramos Álvarez, A. Ribagorda Garnacho. *Avances en criptología y seguridad de la información.* Ediciones Días de Santos, 2004
4. http://www.bvs.sld.cu/revistas/aci/vol11_6_03/aci11603.htm
5. http://www.cert.fnmt.es/tuto1.htm
6. http://www.coresecurity.com/corelabs/projects/research_community/Waissbein_2003.pdf
7. http://www.cryptoforge.com.ar/seguridad.htm
8. http://www.csrc.nist.gov/publications/
9. http://www.cu.ipv6tf.org/pdf/victor_villagra.pdf
10. http://www.dccia.ua.es/dccia/inf/asignaturas/TIC/Curso%202003-2004/Material/TEMA4.pdf
11. http://desarrollo.cenac.ipn.mx/workshop/Documents/24-Articulo%20CMICE2005.doc
12. http://eia.udg.es/~rrecasen/pdf/Criptografia_y_seguridad.pdf

13. http://www.esat.kuleuven.ac.be/

14. http://www.eurologic.es/cifrado/simetric.htm

15. http://www.fombella.org/astic2005/Volumen%20II%20PDF/071.pdf

16. http://it.unex.es/sypi/articulo/cripto.pdf

17. http://www.ietf.org/rfc/rfc3566.txt

18. http://www.matesco.unican.es/talleres_matematicas/transparencias/transparencias-jaime.pdf

19. http://www.noticias.com/notaprensa/16-09-2005/tlr/espana-se-acerca-al-millon-usuarios-firma-digital-709.html

20. http://office.microsoft.com/es-es/assistance/HP052495553082.aspx

21. http://www.onnet.es/06041.htm

22. http://paginas.fe.up.pt/~jvv/Assuntos/encrypt/full.html

23. http://premium.vlex.com/doctrina/REDI_Revista_Electronica_Derecho_Informatico/Uso_Firma_Electronica_Administracion_Comentario_Resolucion_General_345-99_Comision_Nacional/2100-107494,01.html

24. http://proyectos.gpul.org/charlas/criptografia2004/cifrado.pdf

25. http://www.rediris.es/cert/proyectos/iris-pca-copia/gti-pca/tras-JT01/Librerias_Criptografica.ppt#12

26. http://www.seriema-systems.com/index.php?action=filegetter&file=wp-ss-sec-01.pdf

27. http://webpages.ull.es/users/cryptull/TallerCripto3/Apuntes/CriptografiaClaveSecretaCifFlujo.pdf

# Anexo A.

## Postulados de pseudoaleatoriedad de Golomb:

1. En cada periodo de la secuencia considerada, el número de unos tiene que ser aproximadamente igual al número de ceros. Más concretamente, la diferencia entre uno y otro no debe exceder la unidad.
2. En cada periodo de la secuencia considerada, la mitad de las rachas – del número total de las rachas observadas – tiene la longitud 1, una cuarta parte tiene longitud 2, una octava parte tiene la longitud , etc. Al mismo tiempo, para cada una de las longitudes anteriores habrá el mismo número de rachas de ceros que de unos.
3. La autocorrelación AC(k) fuera de fase es constante para todo valor de k.

## Condiciones de secreto perfecto.

Shanon definió sus condiciones de secreto perfecto partiendo de dos hipótesis básicas:

1. La clave secreta se utilizará solamente una vez, a diferencia de lo que sucedía en los métodos clásicos, en los que la clave era fija.
2. El enemigo criptoanalista tiene acceso sólo al criptograma; luego está limitado a un ataque sobre el texto cifrado únicamente.

Basadas en estas dos hipótesis, Shannon enunció que un sistema criptográfico verifica las condiciones si el texto claro X es estadísticamente independiente del criptograma Y, es decir, la probabilidad que una variable X toma el valor x es la misma con o sin conocimientos de valor tomado por la variable aleatoria Y. En términos más sencillos esto equivale a decir que la información sobre el texto claro aportada por el crioptosistema es nula.

Asimismo Shannon determinó la menor cantidad de clave necesaria para que pudieran verificarse las condiciones de secreto perfecto. La longitud de la clave K tiene que ser al menos tan larga como la longitud del texto claro M:

$$K \geq M.$$

### Cifrado Vernam.

El método Vernam fue utilizado durante la segunda guerra mundial por espías de diversas nacionalidades. Este cifrado emplea un alfabeto binario, pues inicialmente se utilizó para las comunicaciones telegráficas haciendo uso del código Baudot ( cinco dígitos binarios por carácter alfabético). La operación aritmética es una suma módulo 2, y la clave es una secuencia binaria perfectamente aleatoria de la misma longitud que el texto claro.

Para recuperar el mensaje original se suma nuevamente al criptograma la secuencia aleatoria, ya que adición y sustracción coinciden en la aritmética módulo 2. La originalidad del procedimiento Vernam radica en que la clave se utiliza solamente una vez (one-time pad). Como la suma módulo 2 con una secuencia aleatoria ofrece un perfecto enmascaramiento del contenido y estadística del texto claro, el cifrado Vernam verifica las condiciones de secreto perfecto de Shannon [*Anexo A, Condiciones de secreto perfecto*]. Dentro del panorama criptográfico actual, el cifrado Vernam es el único procedimiento incondicionalmente seguro o procedimiento con seguridad probada matemáticamente.

## Anexo B.

### Creación del vector inicial a partir de la contraseña.

La generación del vector inicial para el cifrado AES en modo CBC se realiza utilizando la clase de la librería Java SecureRandom para generación de la secuencia, empleando la cadena de bits inversa obtenida de la contraseña como la semilla. Como algoritmo para creación del vector ha sido elegido el algoritmo SHA1 (SHA1PRNG en librería de Java).

Una función hash (función picadillo) es una función computable que aplica a un mensaje m de tamaño variable, una representación de tamaño fijo del propio mensaje: H(m), que es llamado su valor hash :

$$H : M \rightarrow M,\ H(m) = m'$$

La función hash SHA-1 (Secure Hash Algorithm) es una de las funciones hash más utilizadas. El único ataque que se conoce contra este tipo de funciones es el de la investigación exhaustiva.

# Anexo C.

## Manual de usuario.

## 1. Instalación

La aplicación INERCIPHER no necesita la instalación alguna. Se presenta en un archivo comprimido INERCIPHER.zip y para poder utilizarla es necesario descomprimir este archivo en un directorio, abrir la carpeta INERCIPHER y ejecutar el archivo INERCIPHER.bat realizando doble clic sobre el último. Como consecuencia se abre la pantalla de interfaz gráfica de la aplicación.

*Figura 1. La pantalla principal de la aplicación.*

## 2. Inicialización de aplicación.

Antes de realizar las operaciones disponibles (cifrado/descifrado de archivos y/o verificación de la integridad de los documentos) es necesario elegir el archivo en concreto. A partir de este momento van apareciendo las pantallas interactivas con la información necesaria para seguir paso a paso.

## 3. Elección del fichero.

Para elegir un fichero determinado se abre la opción de barra de menú **Fichero** y se realiza "clic" con el botón izquierdo de ratón sobre la opción **Abrir.**

*Figura 2. Menú inicial.*

Como el resultado aparece la ventana con la lista de directorios. Al realizar la elección el documento se acepta y esta confirmación da paso a la siguiente etapa de ejecución de aplicación – elección de la opción.

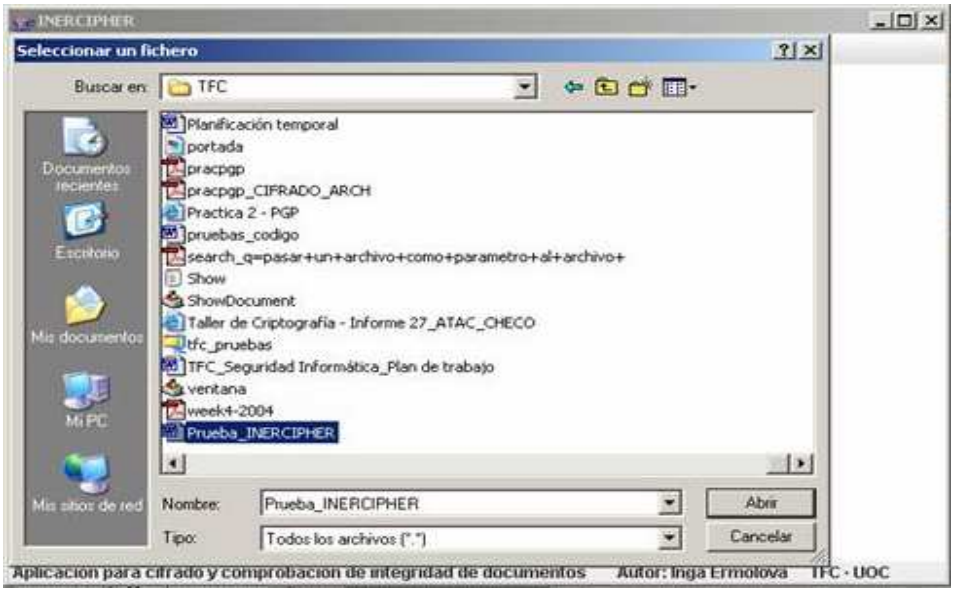


*Figura 3. Selección de un archivo..*

## 4. Opción CIFRADO.

Para realizar cifrado o descifrado de los documentos hay que elegir la opción deseada en la lista con etiqueta CIFRADO (opción **Cifrar** o **Descifrar**).

*Figura 4. Elección de la opción cifrado/descifrado.*

Si la opción elegida es Cifrar, pero ya existe el fichero que contiene el texto cifrado del documento seleccionado, el proceso no se realizará.

*Figura 5. El fichero cifrado ya existe.*

Para descifrar un archivo el archivo elegido ha de tener la extensión ***.cifr*** en caso contrario aparece el mensaje de error. Con la contraseña equivocada, pero que tiene la longitud adecuada el proceso de descifrado se realizar igualmente, sin embargo los datos que se guardarán en el archivo descifrado serán incorrectos.

*Figura 6 .Elección de archivo sin extensión .cifr para descifrar*

Al seleccionar una opción aparece la pantalla para entrada de contraseña (ver punto 6. Contraseña de este documento).

Mientras se ejecuta el proceso se aparece la pantalla con mensaje conforme que se está realizándose la operación elegida que confirma el funcionamiento correcto de la aplicación. Los archivos de tamaño importante (los archivos de sonido, los archivos de imágenes con animación etc.) pueden tardar a cifrarse hasta varios minutos.

*Figura 7. Realización de la operación elegida.*

En caso de cifrado de los archivos al acabar el proceso se puede eliminar el archivo con texto claro (opción recomendada), sin embargo el usuario ha de valor la posibilidad de guardar las copias de archivos que serán cifrados en un dispositivo de memoria extraíble. Si no se realiza este proceso antes proceder con el cifrado, entonces sería conveniente guardar ambos archivos (con el texto claro y cifrado). De ese modo al finalizar la ejecución de la aplicación INERCIPHER se podrá realizar las copias de archivos con texto claro y su eliminación de manera conveniente.

*Figura 8 .Eliminación del fichero con texto claro.*

La finalización correcta del proceso se avisa con un mensaje.

*Figura 9. Finalización de la operación elegida.*

Los ficheros creados se guardan en el mismo directorio, que el fichero seleccionado para realizar las operaciones.

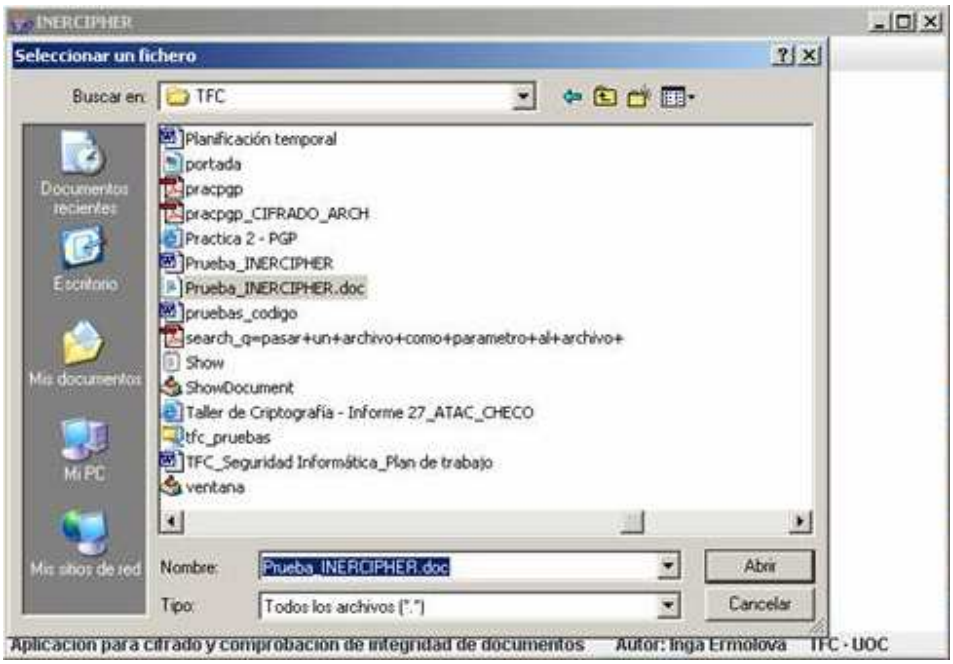


*Figura 10. Fichero Prueba_INERCIPHER.doc.cifr guardado en el mismo directorio que el fichero Prueba_INERCIPHER.doc*

El archivo con texto en claro Prueba_INERCIPHER.doc contenía la información siguiente:

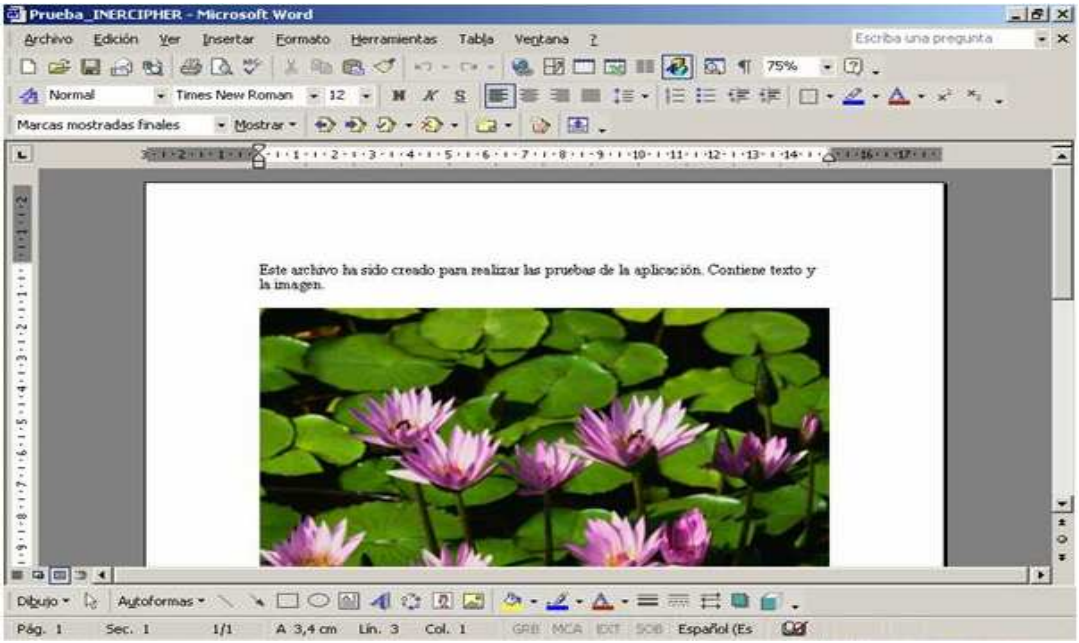


*Figura 11.Archivo con texto claro.*

El documento con la información cifrada se puede visualizar con cualquier editor de texto, por ejemplo *Bloc de notas* de Windows y ha de tener el aspecto siguiente:

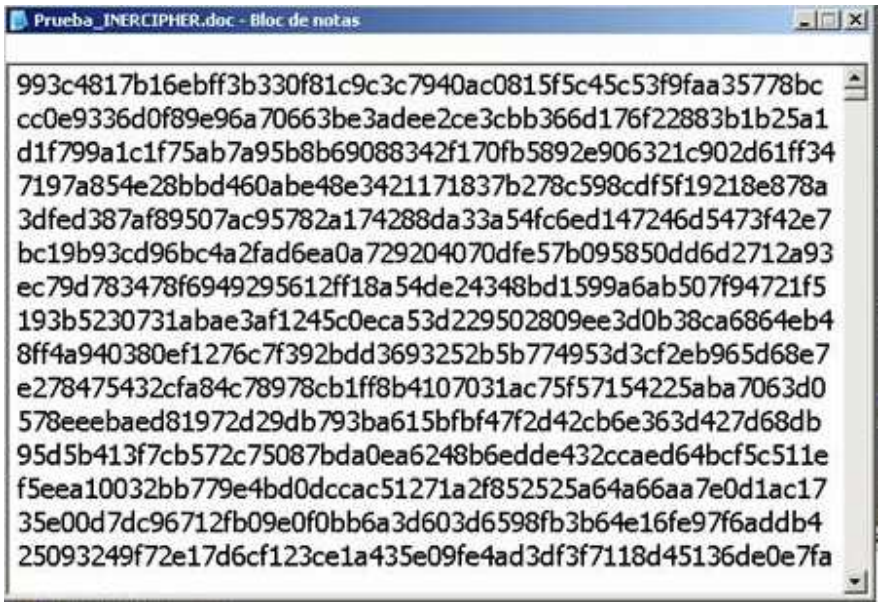


*Figura 12. Documento cifrado (extensión .cifr)*

La selección de opción **Salir** en el menú CIFRADO permite volver a la pantalla inicial para comenzar el proceso de elección de documento desde principio.

Para cifrar/descifrar otro archivo el proceso se repite exactamente, comenzando por la elección de archivo.

## 5. Opción INTEGRIDAD.

El proceso de la generación de código MAC y de la verificación de Integridad de documento se inicia de la misma manera como en caso de cifrado.

En la pantalla de elección de la opción deseada en este caso se abre la lista de opciones con la etiqueta **INTEGRIDAD.**

Al realizar la selección de opción deseada aparece la pantalla para entrada de contraseña. Igualmente como en caso de cifrado/descifrado mientras el proceso está en curso, aparece la pantalla con el mensaje de aviso (ver *Figura 7*.)

*Figura 13. Verificación de integridad*

La realización correcta de operación se confirma con el mensaje adecuado:

- En caso de generación de código MAC

*Figura 14. Generación de código MAC.*

· Comprobación de integridad en caso que el mensaje no ha sido modificado

*Figura 15. Documento integro.*

· Comprobación de integridad en caso que el mensaje no ha sido modificado

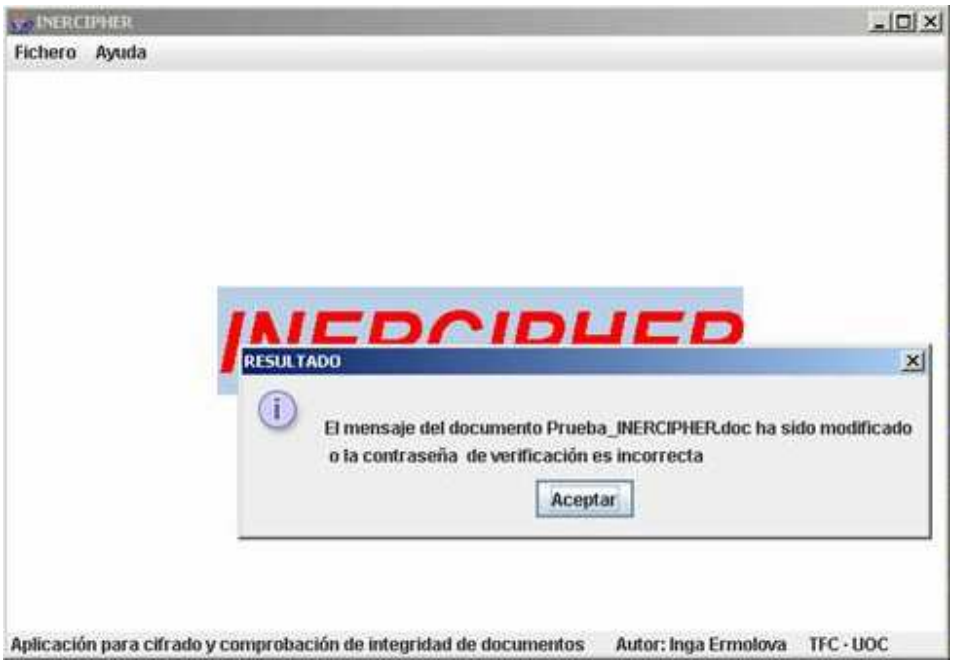


*Figura 16. Documento modificado.*

En caso que se intenta verificar la integridad del documento sin que este último tenga el código MAC generado previamente aparece el mensaje de error y el proceso debe ser repetido desde punto de elección de archivo.

*Figura 17. No existe el código MAC del documento seleccionado para la verificación.*

La selección de opción **Salir** en el menú INTEGRIDAD permite volver a la pantalla inicial para comenzar el proceso de elección de documento desde principio.

Para generar el código MAC o verificar la integridad de otro archivo el proceso se repite exactamente, comenzando por la elección de archivo.

## 6. Contraseña

La entrada de contraseña se realiza a través de la pantalla que dispone de campo para los caracteres de contraseña. Al escribir el número suficiente de caracteres en caso que usuario quisiera seguir ejecutando la aplicación debe aceptar la contraseña.

*Figura 18. Pantalla para introducción de contraseña.*

En caso de utilizar la contraseña más corta, aparece el mensaje de error.

*Figura 19. Entrada de contraseña que contiene menos de 12 caracteres.*

La aceptación de este mensaje provoca la aparición de nuevo la pantalla para entrada de contraseña (ver Fig. 18).

Hay que tener en cuenta, que la aplicación por razones de seguridad no dispone de un mecanismo para almacenar las contraseñas para poder recodarlas al usuario, lo que significa que el mismo usuario a de procurar de no olvidar la contraseña.

La contraseña que aparece en el campo de **password** está enmascarada (también por razones de seguridad), por lo cual se recomienda proseguir con el proceso de cifrado o de la verificación solo en caso que el usuario está seguro de introducir la contraseña correctamente. En caso contrario, al no disponer de la contraseña adecuada, sobre todo durante el proceso de cifrado, se puede perder la información guardada en el archivo. En caso de verificación de la integridad de documentos pueden haber los falsos negativos (el documento se considerará como modificado – no íntegro – erróneamente).

## 7. Opción Ayuda

Opción **Ayuda** de menú principal facilita al usuario la información básica sobre el funcionamiento de la aplicación sin necesidad de buscarla en el Manual. Consiste de la opción **FAQ**, que contiene la lista de preguntas más frecuentes y la opción **Salir** para finalizar la aplicación.

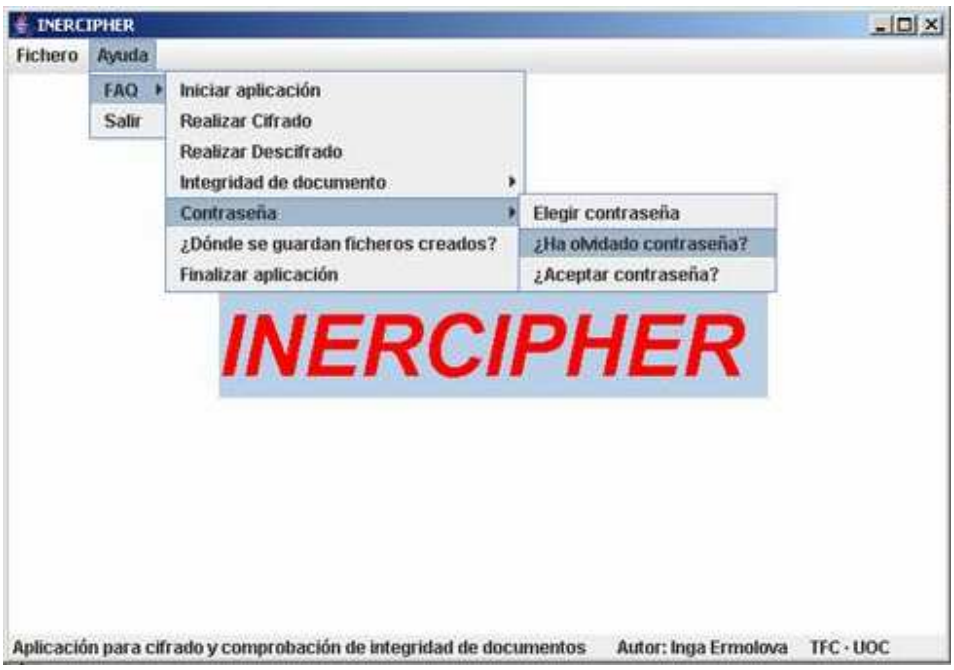


*Figura 20. Opciones de ayuda.*

Al seleccionar una de las opciones aparece el mensaje informativo correspondiente

*Figura 21. Mensaje de ayuda.*

En caso de no resolver problema, se recomienda buscar la información necesaria en el **Manual de usuario**.

## 8. Finalizar la aplicación

Para salir de la aplicación hay que seleccionar la opción **Salir** de la lista de las opciones de Menú principal **Fichero** ver Fig. 2) o **Ayuda** (ver Figura 20).