# IMPLEMENTACION DE UN CIFRADOR BASADO EN UNA PERMUTACIÓN PSEUDO-ALEATORIA

**Hernando Castañeda Marín**

**UNIVERSIDAD AUTONOMA DE BUCARAMANGA**
**INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY**
**BUCARAMANGA**
**2005**

# IMPLEMENTACION DE UN CIFRADOR BASADO EN UNA PERMUTACIÓN PSEUDO-ALEATORIA

**HERNANDO CASTAÑEDA MARÍN**

**Tesis presentada para optar**
**Al Título de Maestro en Ciencias Computacionales.**

**Director**
**JUAN CARLOS MARTINEZ**

**UNIVERSIDAD AUTONOMA DE BUCARAMANGA**
**INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY**
**BUCARAMANGA**
**2005**

**Notas de aceptación**

_____

_____

_____

_____
Presidente del jurado

_____
Jurado

_____
Jurado

Bucaramanga, Mayo del 2005

*Como siempre, a Dios a quien acudí cuando*

*Tuve que requerir de su infinito amor,*

*pido perdón a El y a quienes he ofendido.*

*Finalmente manifestar Que la ciencia es*

*significativamente extensa como mujeres hay en el mundo*

*Quisiera saber todo pero solo debo convivir y amar a una.*

*Ella se llama Stella.*

***Hernando***

## AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

A la Universidad de Pamplona y específicamente al profesor Juan Carlos López Carreño Decano de la Facultad de Ciencias Naturales por su amabilidad en permitir mis desplazamientos semanales a las sesiones satelitales y clases presénciales en la maestría en ciencias computacionales.

Agradecer al los directivos de la Universidad Autónoma de Bucaramanga por la muestra de afecto, comprensión y la calidad organizacional que me mostraron durante mi permanencia en los claustros universitarios.

Agradecer al ingeniero Juan Carlos Martínez y personal vinculado a Laboratorio de Computo Especializado por la forma de enseñarme a trabajar en grupo con lideres invisibles y en su misión Investigativa me hace sentir una persona competente y cada día más autosuficiente.

Quiero agradecer al Doctor **Wilson Zúñiga Galindo** por sus declaraciones y definiciones conceptuales sobre las metas investigativas y actitudes personales para llegar a ser un investigador íntegro en los diferentes campos de la ciencia.

Finalmente quiero agradecer a mis compañeros de la segunda promoción de la maestría en ciencias computacionales y expresarles que al menos sus nombres nunca podrán olvidarse.

**CONTENIDO**

## LISTA DE FIGURAS

**Pág.**

## LISTA DE ANEXOS

**Pág.**

## RESUMEN

TITULO: IMPLEMENTACION DE UN CIFRADOR BASADO EN UNA PERMUTACIÓN PSEUDO-ALEATORIA*

AUTOR: HERNANDO CASTAÑEDA MARÍN**

DESCRIPCIÓN

En nuestra opinión los fundamentos de la criptografía son paradigmas, aproximaciones y técnicas utilizadas para conceptuar, definir y proveer soluciones en los problemas naturales de la criptografía.

Los diseñadores de Yarrow establecían como un punto en el futuro la probabilidad de ver un nuevo cifrador estándar el AES. En este proyecto el diseño básico de Yarrow-160 es laboriosamente acomodado a la construcción de un cifrador haciendo uso de una simple permutación. Se ha demostrado teóricamente que el resultado de los cifrados, son seguros cuando la permutación es aleatoria o seudo-aleatoria. Lo anterior remueve la necesidad de almacenar o generar una multitud de permutaciones.

Se describe el diseño de Yarrow, una familia de generadores criptográficos de números seudo-aleatorios. Se describe el concepto de un PRGN con primitivas criptográficamente diferentes y su principal diseño desarrollado mediante Yarrow.

Hemos realizado la construcción de un cifrador utilizando una permutación referenciado y utilizando el software de Yarrow –160 y se tienen que realizar pruebas para demostrar una significativa seguridad en los rasgos de esta herramienta.

En la práctica se espera encontrar cualquier debilidad tanto en el esquema como en su construcción para llegar a estimar probablemente la entropía de éste, mediante un criptoanálisis.

Para mostrar las posibles debilidades se llevan a cabo pruebas estadísticas, aunque no son medidas definitivas, se proponen pruebas de hipótesis, para medir la calidad de las secuencias producidas por el PRGN.


---

* Tesis de grado


** Universidad Autónoma de Bucaramanga. Instituto Tecnologico y de Estudios Superiores de Monterrey, Juan Carlos Martínez.

# INTRODUCCIÓN

Para diseñar un sistema de seguridad se tiene que seleccionar los servicios de seguridad que hayan sido previstos y luego seleccionar los mecanismos que se implementarán por ejemplo: La encriptación, las firmas digitales, el control de acceso, la integridad de los datos y la autenticación.

La operación primitiva empleada por la criptografía es la encripción. La encripción es una computación especial sobre mensajes, que los convierte en una representación que sólo tiene sentido significado para un receptor específico. Así que alguna forma de encripcíon es necesaria para enviar y almacenar datos sobre redes de comunicación inseguras.

La encriptación es una función matemática, los algoritmos de encriptación deben tener la propiedad de que los datos originales puedan ser recuperados a partir de los datos encriptados, si el valor de la clave utilizada es bien conocido. Las entradas al algoritmo de encriptación son referidas como texto plano, la salida del algoritmo es denominado texto cifrado o criptograma.

Las claves deben ser mantenidas en secreto, el criptograma estará comprometido si un intruso puede deducir la clave analizando la información pública disponible. La fortaleza de un criptosistema es medida por la dificultad, usualmente medido en el número de operaciones elementales, para determinar dicha clave.

Hay dos clases de criptosistemas, simétricos y asimétricos, llamados también de encriptación pública. Un criptosistema simétrico utiliza la misma clave en ambos extremos de un canal de comunicación, o una clave que es fácilmente derivada de otra, en caso de hacer uso de dos claves. En los criptosistemas asimétricos cada usuario tiene dos claves, una pública y una privada, la cual no es revelada.

La encriptación presenta algunas limitaciones prácticas visibles como son: Los mensajes de entrada a los algoritmos no cuentan con protección, el intercambio de claves, se realiza sobre los mismos canales en que se transmiten los datos, la encriptación no provee protección contra las modificaciones, esta puede ser detectada incluyendo como parte de los datos encriptados un número de bits de chequeo.

El trabajo de tesis tiene como propósito desarrollar y analizar una implementación práctica del cifrador con base en el esquema propuesto por Even y Mansour[1]. El esquema corresponde a un cifrador de bloque con base en el uso de una permutación en la operación de encriptación y su correspondiente función inversa para el proceso de desencriptación.

La clave $K$ que consiste en dos subclaves $K_1$ y $K_2$, las que son producidas mediante un generador de números seudo-aleatorios cuyas entradas son muestras impredecibles y sus salidas secuencias seudo-aleatorias.

Para medir la confiabilidad de la permutación de la secuencia resultante de la operación XOR previa, se realizaron pruebas de hipótesis haciendo uso del estadístico $\chi^2$ para diferentes patrones de combinaciones de $\{0,1\}$ referenciadas en dichas secuencias.

Para lograr este propósito se utilizó la herramienta computacional Yarrow-160, un software en visual C++ disponible por Counterpane System estudiado y analizado hasta enriquecerlo con las rutinas de cifrado y descifrado para el criptosistema propuesto y con base en el esquema de Even y Mansour. Adicionalmente se realizó una rutina que permite hacer pruebas estadísticas mediante la determinación de los valores calculados de $\chi^2$ para los diferentes patrones combinatorios de las secuencias.

En el capitulo de cifradores de secuencias se explicaron algunos fundamentos de la criptografía mediante estudio de técnicas matemáticas relacionadas con la seguridad de la información, la confidencialidad y la integridad de los datos. Se describieron herramientas computacionales como son los cifradores de bloque, los cifradores de secuencias, las secuencias aleatorias y las secuencias seudo-aleatorias, los generadores de números seudo-aleatorios y acumuladores de entropía que son utilizados para proveer seguridad en la información.

La metodología empleada durante la investigación cumplió con los siguientes pasos: Una revisión bibliografíca exhaustiva que terminó con una sustentación publica del anteproyecto, una revisión bibliográfica del artículo de Even y Mansour [1] y una revisión computacional de criptosistemas y generadores de números seudo-aleatorios como requisitos previos para una sustentación pública de un avance de tesis; en la fase final se desarrollo el criptosistema enriqueciendo el software del PRNG Yarrow-160 y la aplicación de prueba estadística con el afinamiento del presente informe final.

Este trabajo es un práctica académica que pone a prueba las fortalezas y debilidades de un software propuesto por Counterpane System , y permite que sea enriquecido mediante la construcción de un cifrador basado en una simple permutación que , como implementación práctica permite pruebas tanto en su complejidad computacional como en su seguridad. La realización de un criptoanálisis y la realización de diferentes ataques criptográficos harán posible trabajos futuros en el campo de la criptografía y seguridad computacional.

# 1. CIFRADORES DE SECUENCIA

Los cifradores son clasificados como cifradores de bloque y cifradores continuos, se han utilizado durante siglos como guardián de la seguridad militar y la comunicación diplomática. Así, la criptográfia encontró una aplicación directa, al permitir proveer de confidencialidad a toda comunicación.

La criptografía no sólo provee de confidencialidad a un sistema, en general reduce considerablemente los riesgos de seguridad en cuatro formas principales: confidencialidad, autentificación, integridad y control de acceso.

Un sistema criptográfico es aquel cuyo propósito principal es manipular información de manera secreta. Con manipular información, se refiere a transferencia, almacenamiento, difusión, codificación.

## 1.1 DOMINIOS Y CODOMINIOS DE ENCRIPTACIÓN.

A denota un conjunto finito llamado alfabeto de definición, por ejemplo $A = \{0,1\}$ es un alfabeto binario, es usado frecuentemente como un alfabeto de definición. Cualquier alfabeto puede ser codificado en términos del alfabeto binario.

M denota un conjunto llamado espacio del mensaje, este consiste de una cadena de símbolos de un alfabeto de definición. Un elemento de M es llamado mensaje de texto plano o simplemente texto plano. Ejemplo: M consiste en cadenas binarias, texto en ingles, código de computación etc.

C denota un conjunto llamado espacio del criptograma, éste consiste en una cadena de símbolos de un alfabeto de definición, el cual puede ser diferente del alfabeto de difinición de M. Un elemento de C se llama texto cifrado.

## 1.2 TRANSFORMACIONES DE ENCRIPTACIÓN Y DESENCRIPTACIÓN

K denota un conjunto llamado espacio de claves. Un elemento de K es llamado clave. Cada elemento $e \in K$ determina únicamente una biyección de M a C denotado por $E_e$, que es una función de encriptación o una transformación de encriptación. Note que $E_e$ tiene que ser una biyección, si el proceso es el de invertir como un mensaje de texto plano único, recuperado por cada texto cifrado diferente.

Para cada $d \in K, D_d$ denota una biyección de C a M. $D_d$ Es una función de desencriptación o transformación de desencriptación.
El proceso de aplicar la transformación $E_e$ a un mensaje $m \in M$ se considera como encriptación de m.

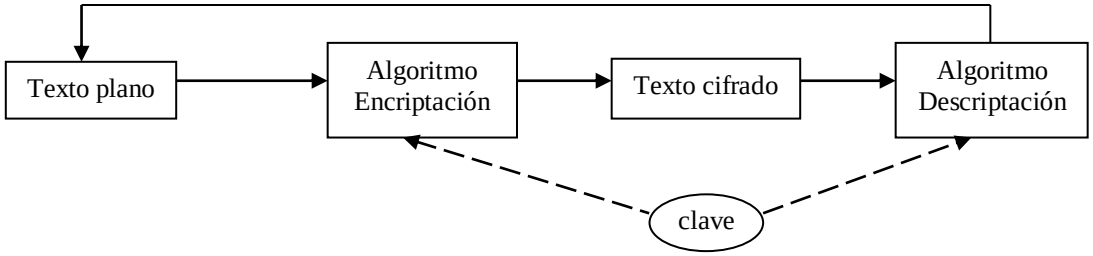
El proceso de aplicar la transformación $D_d$ a un texto cifrado $c \in C$ se considera una desencriptación de $c$.

El esquema de encriptación consiste en un conjunto de transformaciones de encriptación $\{E_e : e \in K\}$ y corresponden a un conjunto de transformaciones de desencriptación $\{D_d : d \in K\}$ con la propiedad que por cada $e \in K$ hay una clave única $d \in K$ tal que $D_d = E_e^{-1}$ esto es $D_d(E_e(m)) = m$. Un esquema de encriptación, algunas veces se llama cifrador.

Las claves $e$ y $d$ en la anterior definición se refieren a un par de valores de claves y algunas veces pueden tener el mismo valor.

La construcción de un esquema de encriptación requiere una entidad seleccionada en el espacio de mensajes, una en el espacio de textos cifrados y una en el espacio de claves tambien un conjunto de transformaciones de encriptación $\{E_e : e \in K\}$ y su correspondiente conjunto de transformaciones de desencriptación $\{D_d : d \in K\}$.

Figura 1. Esquema General de encriptación/desencriptación




## 1.3 CONFIDENCIALIDAD

El esquema de encriptación puede ser usado como se muestra a continuación con el propósito de lograr confidencialidad. Dos partes Alicia y Bob seleccionan e intercambian secretamente un par de claves. En un punto sucesivo en el tiempo, si Alicia desea enviar un mensaje $m \in M$ a Bob, ella computa $c = E_e(M)$ y transmite esto a Bob. Después de recibir c, Bob computa $D_d(c) = m$ y así recupera el mensaje.

La pregunta clave es ¿por qué las claves son necesarias ? (¿ por qué no se selecciona una función de encriptación y su correspondiente función de desencriptación?) teniendo transformaciones que son muy similares pero caracterizadas por medios en los cuales, si algunas transformaciones de encriptación-desencriptación son reveladas y si no tiene que rediseñarse el esquema completo, simplemente se cambia la clave? Esta es una práctica

criptográfica légitima que consiste en cambiar la clave frecuentemente en las transformaciones de encriptación -desencriptación.

Como un análogo físico considera una combinación ordinaria de cerradura reajustada. La estructura de la cerradura está disponible a cualquiera que desee adquirirla, pero la combinación es seleccionada y puesta por los propietarios. Si los propietarios sospechan que la combinación se ha revelado, alguno puede restablecerla fácilmente sin necesidad de reemplazar el mecanismo físico.

## 1. 4 PARTICIPANTES EN LA COMUNICACIÓN

Una entidad o parte es alguien que envía, recibe y manipula información. Una entidad que puede ser una persona o un terminal de un computador.

Un transmisor es una entidad en la comunicación de dos partes, donde ocurre una transmisión legítima de información.

Un receptor es una entidad en la comunicación de dos partes, que es el destinatario intencional de la información.

Un adversario, es una entidad en la comunicación de dos partes que no es ni el transmisor ni el receptor. Entre los varios nombres que son sinónimos del adversario están enemigo, asaltador, antagonista, osado, indiscreto, intruso, y entrometido. Un adversario intentará a menudo jugar el papel de remitente legítimo o de receptor legítimo.

## 1.5 SEGURIDAD

Una premisa en criptografía es que los conjuntos $\mathrm{M}, \mathrm{C}, \mathrm{K}, \{\mathrm{E}_{\mathrm{e}} : e \in K\}, \{D_d : d \in K\}$ son de conocimiento público. Cuando las dos partes desean comunicarse,

seguramente usando un esquema de encriptación, sólo se debe pensar que el secreto confidencial es un par de claves particulares $(e,d)$ que ellos usan y que ellos pueden seleccionar. Se puede ganar seguridad adicional guardando la clase de encriptación y los secretos de transformación de descriptación, pero se tiene que basar la seguridad en el esquema entero. La historia ha mostrado que mantener el secreto de la transformación es muy difícil de hecho.

Un esquema de encriptación se dice que es rompible si una tercera parte, sin conocimiento previo del par de claves, puede sistemáticamente recuperar el archivo de texto desde el correspondiente texto cifrado dentro de alguna estructura apropiada.

Un esquema de encriptación puede romperse probando todas las posibles claves para ver cuál están usando las partes comunicantes, se supone que la clase de función de encriptación es de conocimiento público. Esto se llama una búsqueda exhaustiva del espacio de claves. Sigue entonces que el número de claves (es decir el tamaño del espacio de claves) debe ser significativamente grande, para hacer que esto no sea factible en forma computacional. Un conjunto de requerimientos para un sistema de encriptación son citados en la literatura como Kerckhoffs desiderata [5]. Entre ellos se tiene: Si el sistema no es rompible teóricamente tampoco es rompible en la práctica; el compromiso de los detalles del sistema no debe incomodar a los corresponsales. La clave debe tenerse presente sin notas y fácilmente debe cambiarse el aparato de encriptación, que debe ser portátil y operable por una sola persona, El sistema debe ser fácil, mientras no requiera el conocimiento de una lista larga de reglas

## 1.6 CIFRADORES DE BLOQUE

Definición: Un cifrador de bloque es un esquema de encriptación en el cual se divide el mensaje de texto plano que va a ser transmitido en cadenas (llamadas

bloques) de una longitud fija t sobre un alfabeto A y se tiene que encriptar un bloque al mismo tiempo.

Muchas de las técnicas bien conocidas sobre encriptación de claves simétricas consisten en cifradores de bloque. Las dos importantes clases de cifradores de bloque son cifradores de substitución y cifradores de transposición. Los cifradores de producto combinan a éstos.

Definición: Sea $A$ un alfabeto de $q$ símbolos y $M$ el conjunto de todas las cadenas de longitud $t$. Sea $K$ el conjunto de todas las permutaciones sobre el conjunto. Se define para cada $e \in K$, una transformación de encriptación $E_e$ así:

$E_e(m) = (e(m_1)e(m_2)...e(m_t)) = (c_1c_2...c_t) = c$, Donde, en otras palabras, para cada símbolo en una t-tupla se remplaza (sustituye) por otro símbolo de A de acuerdo con alguna permutación fija.

Para desencriptar $c = (c_1, c_2, ..., c_t)$ computando la permutación inversa $d = e^{-1}$ y $D_d = (d(c_1)d(c_2)...d(c_t)) = (m_1m_2...m_t) = m$, $E_e$ se llama cifrador de sustitución simple o cifrador de sustitución monoalfabética.

El número de sustituciones directas del cifrador es $q!$ y es independiente del tamaño del bloque en el cifrador, en el alfabeto inglés por ejemplo, el espacio de claves es $26! \approx 4x10^{26}$. Son simples sustituciones (cifrador del césar) sobre bloques de tamaño pequeño que proveen una seguridad inadecuada, incluso cuando el espacio de las claves es extremadamente grande. Por observación, en una cantidad modesta de bloques de texto cifrado, un criptoanálisis puede llegar a determinar la clave.

## 1.7 CIFRADORES CONTINUOS

Los cifradores continuos son un importante esquema de cifradores de clave simétrica. Ellos son en sí, cifradores de bloque muy simples que trabajan con bloques de longitud uno. Esto es útil, por el hecho de que la transformación de encriptación puede cambiar para cada símbolo del texto plano que se va a encriptar. En situaciones donde los errores de transmisión son probablemente altos, los cifradores continuos son ventajosos porque no presentan error de propagación.

Pueden usarse cuando los datos tienen que estar procesando un símbolo a la vez. Ejemplo, Cuando los equipos tienen limitación de datos en la memoria o en la memoria temporal [5].

Definición Sea K un espacio de claves para un conjunto de transformaciones de encriptación. Una secuencia de símbolos $e_1 e_2 ... e_i \in K$ se considera una secuencia de claves.

Definición Sea A un alfabeto de q símbolos y sea $E_e$ un cifrador simple de sustitución con bloques de longitud uno. Sea $m_1 m_2 ...$ una secuencia del texto plano y sea $e_1 e_2 e_3 ...$ una secuencia de claves de K. Un cifrador continuo toma una secuencia de texto plano y produce una secuencia de texto cifrado $c_1 c_2 c_3 ...$ donde $c_i = E_{e_i}(m_i)$. Si $d_i$ denota el inverso de $e_i$ , entonces $D_{d_i}(c_i) = m_i$ desencripta la secuencia del texto cifrado.

Un cifrador continuo aplica transformaciones simples de encriptación de acuerdo a la secuencia de claves usadas. La secuencia de claves puede ser generada en forma aleatoria o por un algoritmo que genera la secuencia de claves, desde una secuencia pequeña inicial (llamada semilla) o desde una semilla y símbolos

previos de texto cifrado. Tales algoritmos son llamados generadores de secuencias.

## 1.8 ESPACIO DE CLAVES

El tamaño de un espacio de claves es el número par de claves de encriptación-desencriptación que están disponibles en un sistema cifrador. Una clave es típicamente una forma compacta para especificar la transformación de encriptación. Por ejemplo, un cifrador de transposición de longitud de bloque $t$ tiene $t!$ funciones de encriptación de las cuales se seleccionará sólo una. Cada una simplemente es descrita por medio de una permutación que se llama clave.

Es una tentación relacionar la seguridad del esquema de encriptación con el tamaño del espacio de claves.

Dentro de este sistema, la condición para un esquema de encriptación que sea seguro, es que el espacio de claves sea bastante grande con el fin de evitar búsquedas exhaustivas.

Un cifrador continuo especifica un dispositivo con memoria interna que encripta el j enésimo dígito de $m_j$ de la secuencia de mensajes , dentro del j-enésimo dígito de $c_j$, de la secuencia de texto cifrado, mediante una función que depende tanto de la clave secreta $k$ ,como del estado interno del cifrador continuo en el tiempo j.

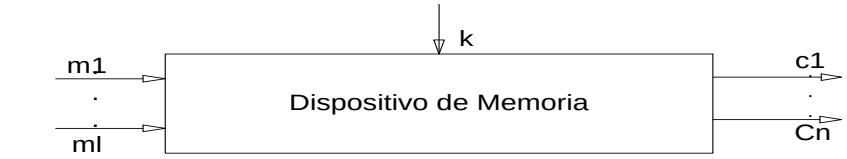
La secuencia $Z^{\infty} = z_0 z_1 ...$que controla la encriptación se llama secuencia de claves o claves de ejecución. El autómata determinístico que produce la secuencia de claves a partir de la clave actual $k$ y el estado interno, se llaman generadores de claves de ejecución o generadores de secuencia de claves.

Un cifrador continuo es periódico si la secuencia de claves se repite un número fijo de veces $d$ ; en caso contrario es no periódico.

Hay dos diferentes aproximaciones de encriptación continuas: método sincrónico y método semi-sincrónico. En el cifrador de secuencias sincrónico el próximo estado depende sólo del estado previo y no de la entrada, así que la sucesión de estados es independiente de la secuencia de los mensajes. La secuencia clave es generada independientemente de la secuencia del mensaje. En el cifrador de secuencias asicrónico cada carácter de la secuencia de clave es derivado de un número fijo $n$ de caracteres cifrados previamente.

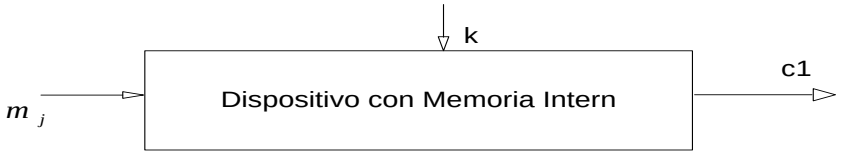
La idea de cifradores continuos semi-sincrónicos se remonta a la antigüedad, la época de Vigeneré en el siglo XVI. Los sistemas de cifradores de auto-claves y cifradores de retroalimentación son ejemplos de cifradores de secuencias semi-sincrónicos aditivos.

Figura 2. La diferencia entre cifradores de bloque y cifradores continuos




$$C = E_k(M)$$

(a) Cifrador de Bloque




$$c_j = E_{xj}(M_j);\ z_j = f(k, c_j - 1)$$

(b) Cifrador Continuo

Hay ventajas y desventajas en los cifradores de bloque y los cifradores continuos aditivos. Los cifradores continuos sincrónicos aditivos tienen la desventaja que el par de caracteres del texto plano y texto cifrado revelan inmediatamente el correspondiente carácter de la secuencia de claves por la cual el carácter del texto plano se va a encriptar.

Esto hace posible varias clases de ataque de recuperación de claves [10] tal como el ataque de correlación y ataques de colisión, ataques de máquina equivalente tal como el basado en el algoritmo de Berlekamp –Massey, y ataques de máquina aproximada tal como el basado en la aproximación lineal. Una de sus ventajas es que la secuencia de claves está variando en el tiempo, lo cual asegura que el mismo carácter de texto plano usualmente corresponda a diferentes caracteres del texto cifrado en diferente tiempo. Esto normalmente oculta algunas propiedades estadísticas del texto plano.

Los cifradores de bloque tienen la desventaja que sus claves no pueden ser cambiadas muy frecuentemente debido al problema de administración de claves. Adicionalmente, el mismo bloque de mensajes corresponde siempre al mismo bloque de texto cifrado, si una clave es seleccionada y fija. Esto puede producir ataques como el ataque diferencial, aplicable en algunos cifradores de bloque.

Una de sus ventajas es que la detección de la modificación de los mensajes puede ser posible debido al hecho de que los mensajes son encriptados bloque a bloque.

## 2. CONSTRUCCIÓN DE UN CIFRADOR BASADO EN UNA PERMUTACIÓN PSEUDO-ALEATORIA

En este capítulo se presenta un cifrador (basado en una permutación Pseudo-aleatoria) desarrollado por S. Even and Y. Mansour. [1]. Este artículo motiva la presente tesis. Se presentaran las ideas fundamentales sin pruebas, siguiendo muy de cerca las de los autores.

Shannon definió un cifrador aleatorio como una colección de permutaciones, escogidas en forma aleatoria una para cada clave [2]. Así un cifrador $C$ consiste en:

1. Un conjunto finito de mensajes y criptogramas $M$ .
2. Un conjunto finito de claves $K$ .
3. A cada clave $k \in K$ esta asociada una permutación $\Pi_k \; \mathrm{M} \to \mathrm{M}$ .

Si cada permutación $\Pi_k$ es escogida en forma aleatoria, con una distribución uniforme, del conjunto de todas las permutaciones hay exactamente un $\#M!$, de estas permutaciones, donde $\#M$ es el número de elementos de M, entonces $C$ es llamado un cifrador aleatorio.

Se nota que en la definición anterior el conjunto de mensajes y criptogramas son iguales, y son identificados con el conjunto $M$.

En caso que $M = \{0,1\}^n$, esto es el conjunto de palabras binarias de longitud $n$, $C$ es llamado un cifrador de bloque.

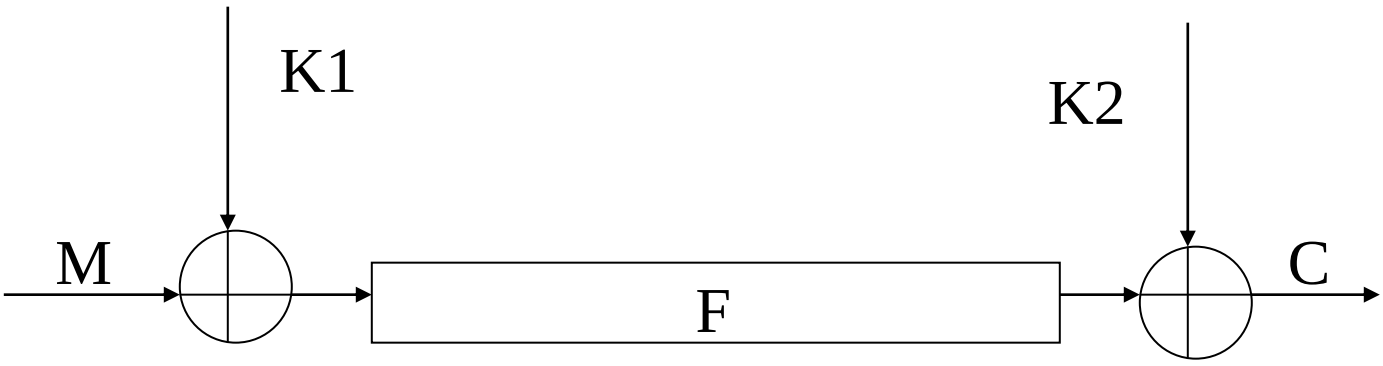
## 2.1 EL CIFRADOR DE EVEN Y MANSOUR

En [1] Even y Mansour proponen la construcción de un cifrador de bloques usando únicamente una permutación seleccionada en forma aleatoria.
El propósito de la presente tesis es desarrollar y analizar una implementación práctica del cifrador de Even y Mansour. Adicionalmente los autores asumen que la permutación se supone que es de acceso público (Como una caja negra), y que cualquiera que desee atacar el cifrador tiene acceso a ella. Los autores "prueban" la seguridad del cifrador propuesto bajo la suposición de que la permutación es aleatoria, o al menos seudo- aleatoria, y que la única vía de accesarla es a través de una caja negra.

Se asume que $M$ y $C$ son palabras binarias de longitud n, esto es que $M$ y $C$ son elementos del conjunto $\{0,1\}^n$.

La siguiente figura ilustra el cifrador a implementar

Figura 3. Esquema de Even y Mansour




Donde $M$ =mensaje, $F$ =permutación $\oplus =$ XOR (bit a bit)
$K = \{k_1, k_2\}$ =Clave, $C$ = criptograma.

También se asume que $F(x)$ es una permutación de $\{0,1\}^n$, y se denota su inversa por $F^{-1}(x)$, y que es "fácil" computar $F(x)$ o, $F^{-1}(x)$, ya sea directamente o usando una caja negra, la cual se denominara "oráculo".

La clave $K$ consiste en dos subclaves $k_1$ y, $k_2$ cada una seleccionada en forma aleatoria del conjunto $\{0,1\}^n$.

Se asume que $K$ es conocida solamente por las partes autorizadas, esto es el emisor y el receptor, y que esta es usada para encriptar mensajes por un largo periodo de tiempo.
El proceso de encriptación puede ser descrito por la siguiente ecuación:

$C = E_K(M) = F(M \oplus K_1) \oplus K_2$, Donde $C$ es el criptograma correspondiente a $\{0,1\}^n$, y $\oplus$ denota la operación XOR (OR exclusivo), bit a bit.

El proceso de descifrado puede ser descrito por la siguiente ecuación:

$$M = D_K(C) = F^{-1}(C \oplus K_2) \oplus K_1$$

## 2.2 DEFINICIONES DE SEGURIDAD Y SUS RELACIONES

En su artículo [1], Even y Mansour modelan al "adversario" como alguien que puede apoderarse por un tiempo determinado del cifrador para cifrar y descifrar mensajes escogidos por él ; que puede usar las permutaciones $F$ y $F^{-1}$, como cajas negras pero que no tiene acceso a la clave $K$.
Even y Mansour modelan la situación anterior usando oráculos, los cuales responden preguntas de naturaleza particular.

El "problema del rompimiento " $CP$, es definido como un intento (por un adversario) de descifrar un criptograma dado $C_0 = E_k(M_0)$ , sin un conocimiento de la clave $K$. Se asume que el algoritmo usado por el adversario tiene acceso a los siguientes oráculos:

1. El $F - oráculo$: Dado $x \in \{0,1\}^n$, el oráculo suministra $F(x)$.
2. El $F^{-1} - oráculo$: Dado $x \in \{0,1\}^n$, el oráculo suministra $F^{-1}(x)$.
3. El $E - oráculo$: Dado $M \in \{0,1\}^n$ , el oráculo suministra $E_k(M)$.
4. El $D - oráculo \quad Co - restringuido$: Dado $C \in \{0,1\}^n$, tal que $C \neq C_0$, el oráculo suministra $D_K(C)$.

En otras palabras se supone que el adversario tiene acceso a una colección arbitrariamente grande de pares $(M,C)$, donde $C$ es el criptograma correspondiente a $M$, generados usando la clave $K$, la cual no es conocida por el adversario y además, que la permutación $F$ es públicamente accesible como una caja negra.

El algoritmo del adversario es exitoso si éste produce $M_0 = D_k(C_0)$. La probabilidad de éxito del algoritmo es la probabilidad de que sea escogida en forma aleatoria *C*, (con distribución uniforme) y el algoritmo produzca $M_0$.

El problema del rompimiento es también conocido como $ataquepormensaje \big/ criptogramaescogidos.$

Ahora se considera el problema de la falsificación existencial, EFP.
El adversario tiene acceso a cuatro oráculos: el $F - oráculo$, $F^{-1} - oráculo$ , $E - oráculo$, $D - oráculo$ no restringido. Este último es definido como sigue: dado cualquier $C \in \{0,1\}^n$ , el oráculo suministra $D_k(C)$.

El objetivo del adversario, en el problema EFP, es encontrar un nuevo par $(M', C')$, con $C^? = E_K(M').$, De esta forma el adversario puede producir un criptograma $C'$ valido para mensajes con "sentido" $M' \neq M_j$ sin conocer $K$. Así el adversario puede enviar mensajes cifrados como si fuera un usuario autorizado.

Definición: sea $f$ una función de los enteros positivos hacia $[0,1]$.

Se dice que $f$ es de forma polinomial despreciable si para todo polinomio hay un $n_0$, tal que si $n \succ n_0$ entonces $f(n) \prec \frac{1}{p(n)}$. [1]

Se asume que el adversario emplea un "algoritmo randómico",[2] cuyo tiempo de corrida es de grado polinomial acotado en $n$, para resolver los problemas $CP$ y $EFP$.

Se dice que un problema es duro, si, para todo algoritmo randómico, la probabilidad de éxito es polinomialmente despreciable. La probabilidad es tomada sobre todas las escogencias hechas en el diseño del cifrador (o sistema) , esto es las escogencias de los $F$, las claves escogidas por el usuario, y los lanzamientos de moneda realizados en el algoritmo del adversario.

Even y Mansour reducen EFP a CP, esto es, ellos muestran que la existencia de un ataque exitoso para el problema CP implica la existencia de un ataque exitoso para el problema EFP.

[1] Comparación asintótica entre dos funciones en el intervalo $[0,1]$ $[5]$

[2] Los algoritmos aleatorios se clasifican de acuerdo a la probabilidad de que retornen una respuesta correcta Pág. 62 $[5]$

Teorema [Corolario 3.2] Si todo algoritmo, con tiempo de corrida polinomial, para el problema EFP tiene una probabilidad de éxito despreciable, entonces todo algoritmo, con tiempo de corrida polinomial, para el problema $CP$ tiene una probabilidad de éxito despreciable.

El resultado principal del artículo de Even y Mansour es el siguiente:

Teorema **[1,*Teorema* 4.4]** La probabilidad de que un algoritmo randómico A resuelva el problema $EFP$, cuando $F$ y $K$ son escogidas aleatoria y uniformemente, está acotada por:

$\boldsymbol{O(\frac{lm}{2^n})}$ Donde l es él numero de preguntas a los oráculos $\frac{E}{D}$, y $m$ es el número de preguntas a los oráculos $\frac{F}{F^{-1}}$.

Como corolario del resultado anterior se obtiene lo siguiente:

Corolario Si $F$ es escogida en forma aleatoria, entonces todo algoritmo randómico con tiempo de corrida polinomial, tiene una probabilidad de éxito despreciable.

Así desde un punto de vista teórico, la probabilidad de "romper" el cifrador de Even y Mansour es despreciable, si F es escogida aleatoriamente.

Si La Familia de permutaciones F son seleccionadas seudo-aleatoriamente y aunque cualquier adversario tiene acceso a los cuatro oráculos, se garantiza que éste no llegará a las entrañas de la caja que implementa F.

La anterior aserción justifica el siguiente teorema **[2,*Teorema* 5.1]** Si F es seleccionado en forma aleatoria y k es seleccionado uniformemente, entonces

para todo algoritmo limitado en grado polinomial para solucionar el problema EFP, la probabilidad del evento es de grado polinomial limitado

## 2.3 LIMITACIONES EN LA CONSTRUCCIÓN

Daemen[4] argumenta que en la prueba de seguridad sobre la complejidad polinomial limitada para un adversario y probadas teóricamente, sufre severas limitaciones y son aparentemente observadas después de la realización de un criptoanálisis. Asegura que tradicionalmente la efectividad de éste se mide en el proceso de búsqueda en el espacio de claves. Establece como causas de dichas limitaciones la aproximación teórica de la complejidad y adicionalmente ante la ejecución de criptoanálisis elementales del esquema, se alcanzan límites superiores en la seguridad, motivando una evaluación realista de la construcción del esquema propuesto por Even y Mansour.

Para el criptoanálisis del esquema las entradas son claves de ***2n*** bits y para una búsqueda exhaustiva requiere $\mathbf{2^{n-1}}$ aplicaciones de ***F*** dentro del espacio de claves.

**2.3.1 Ataque de texto plano conocido.** El criptoanálisis hace referencia a dos pares textos planos conocidos. Sea La operación en el ámbito de bits XOR entre los dos bloques $X_a$ y $\mathrm{X_b}$ se denota por $(X_{a,}C_a)$ y $(X_b,C_b)$. El ataque consiste en calcular $\Delta W = F(V) \oplus F(V \oplus \Delta M)$ para todos los posibles valores de $V$. Si $V = M_a \oplus K_1$ o $V = M_b \oplus K_1$ se cumplirá que $\Delta W = \Delta C$. Si $F$ es aleatorio en función de los pocos valores de $V$ se encuentra para estos valores almacenados. Los valores erróneos de $V^{'!}$pueden fácilmente ser descartados, chequeando que $F(V^!) \neq M_a \oplus K_2$ y $F(V^!) \neq M_b \oplus K_2$, cuando el corrector valor de $V$ se conoce por dos subclaves que pueden determinarse fácilmente. Este ataque toma en

promedio $2^{n-1}$ aplicaciones de $F$ .Aunque reduce la longitud de la clave efectiva a $n$ bits.

**2.3.2 Ataque de texto plano seleccionado.** El criptoanálisis selecciona $K$ pares de texto plano $M_i, M_i^!$ con $M_i \oplus M_i^! = \Delta M$ una constante para toda $i$. Él obtiene los correspondientes textos cifrados $C$. Para $K << 2^n$ hay cerca de $K$ diferentes valores. La velocidad de búsqueda para $\Delta W = \Delta C_i$ para algún valor de $i$ por un factor de $K$ es comparado con el conocido ataque de archivo plano. La magnitud de $K$ está limitada por los requerimientos de memoria de orden $2nK$ bits.

En ausencia de restricciones de memoria hay un óptimo de memoria de $K$ cercano a $2^{\frac{n}{2}}$. En este caso el número esperado de $F$ evaluaciones más encriptaciones es alrededor de, reduciendo la longitud de la clave efectiva a $\frac{n}{2}$ bits .

En él cálculo de la complejidad la permutación $F$ es considerada por ser aleatoria.

Sin embargo, las propiedades no aleatorias de $F$ pueden probablemente ser usadas para la velocidad del ataque. Sin embargo ambos ataques pueden ser paralizados.

Como ejemplo sea $\phi$ un cifrador de bloque tipo Even Mansour con una longitud por bloque de 64 y longitud de claves de 128. Supóngase que un procesador $F$ es un chip que computa F en 100 microsegundos. El desempeño de un criptoanálisis con un ataque archivo de texto seleccionado ( 2 Mbyte) $K \approx 10^5$ de $\phi$ usando una máquina paralela con un procesador F de $10^6$ que recupera la clave en pocas horas.

El conocimiento de un modelo matemático por parte del adversario acerca de la complejidad teórica no implica el conocimiento acerca de la longitud del bloque para que se piense en un esquema seguro.

## 2.4 POTENCIALIDAD DE LA PERMUTACIÓN ALEATORIA PÚBLICA

Even y Mansour[1] probaron que mediante la utilización de una simple permutación seudo-aleatoria $P$ obtenida a partir de una permutación aleatoria publica $F$ sobre $\{0,1\}^n$ y un enmascaramiento de confidencialidad $S$ con $n$ *bits* y formalizada mediante la ecuación: $P(x) = F(x \oplus S) \oplus S$, se pueden construir muchas permutaciones seudo-aleatorias independientes $P_1, P_2, ...P_m$ sobre $\{0,1\}^n$ desde el mismo recurso criptográfico, esto es, una simple permutación aleatoria $F$ sobre $\{0,1\}^n$ y $n$ bits de confidencialidad.

## 2.5 VENTAJAS EN SU CONSTRUCCIÓN

Los análisis realizados por Even Y Manssour [1] y confirmadas por Daemen [4] acerca de las características sobresalientes en la construcción de su cifrador de bloque y que específicamente son: la consideración establecida para el operador de encriptación interno $F$ el cual es fijo y público; y la formalización de su esquema de encriptación. Adicionalmente la demostración sobre la efectiva longitud de la clave $n - lg\, l - lg\, m$ en donde al adversario se le permite realizar $l$ llamadas al oráculo de encriptación y desencriptación y m llamadas al oráculo de la permutación $F\, o\, F^{-1}$.

Como ejemplo de motivación de esta implementación está la construcción de DESX introducida por Rivest y que permite proveer información económica acerca de la clave en DES.

# 3. CONSTRUCCIÓN DEL CIFRADOR

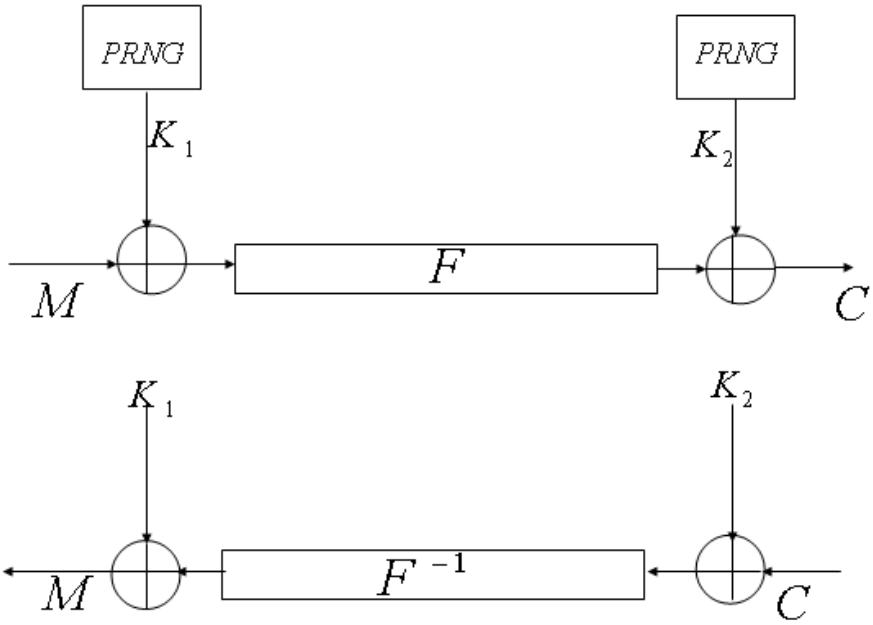
A continuación se explicarán los mecanismos utilizados en la implementación del criptosistema ver fig 4, con base en este esquema propuesto , se oberva que se generarán claves criptográficas aleatorias $K_1$ y $K_2$ utilizando un PRNG , al cifrar se requiere la permutación de la secuencia al operar $K_1 \oplus M$ y para descrifrar la respectiva inversa de la secuencia al operar $C \oplus K_2$. Para mostrar confiabilidad en la implemetacion se hace un análisis estadístico para medir la calidad de las secuencias producidas por el generador de números seudo aleatorios .
En la figura 4 se ilustra el cifrador que se ha de implementar.

Inicialmente se explicarán los mecanismos del generador de números seudo-aleatorios PRNG pero el propósito de utilizar Yarrow-160 para generar las claves se justifican por ser una utilidad para la acumulación de entropía.

Un número aleatorio es un número que no puede ser predecible para un observador antes que éste sea generado. Si un número está en el rango $0...2^{n-1}$, un observador no puede predecir el número con probabilidad superior a $1/2^n$.

Un PRNG contiene un estado interno que es usado en determinados instantes para generar salidas seudo-aleatorios. Este estado guarda confidencia y controla significativamente los procedimientos. Generalmente los números aleatorios son difíciles de generar especialmente en computadores que están diseñados para ser determinísticos, entonces se tiene que recurrIr a los números seudo-aleatorios.

Figura 4. Esquema implementado en el proceso de ciframiento y desciframiento




Donde $M$ =mensaje, $F$ =permutación $\oplus =$XOR (bit a bit),PRNG generador de números pseudo-aleatorios, $K = \{k_1, k_2\}$ =Clave, $C$ = criptograma.

## 3.1 HERRAMIENTA CRIPTOGRÁFICA

Un generador de números seudo-aleatorios es un mecanismo criptográfico para procesar entradas impredecibles y generar salidas seudo-aleatorias. Si se diseña, implementa y utiliza apropiadamente, cualquier adversario con enormes recursos computacionales no será capaz de predecir una secuencia de la salida del PRNG.

El uso de un mecanismo criptográfico llamado generador de números seudo-aleatorios (PRNG Yarrow-160) es quien genera estos valores. Existen valores normalmente coleccionados de procesos físicos, interacciones de usuarios con las maquinas y otros externos, estos son procesos duros de predecir y reciben el nombre de fuentes de entropía,

Un PRNG puede ser visualizado como una caja negra. Un flujo de entrada con todas las medidas internas (muestras), especificadas por el diseñador y Un flujo de salida de números aleatorios generados por el PRNG mediante la creación de un estado no adivinable.

Un adversario puede comprometer el estado interno mediante el uso de un modelo muy bueno que produzca valores impredecibles, utilizando las muestras de entrada del PRNG y con un gran manejo de poder computacional, trate de adivinar el estado interno del PRNG.

El PRNG posee cuatro componentes principales:

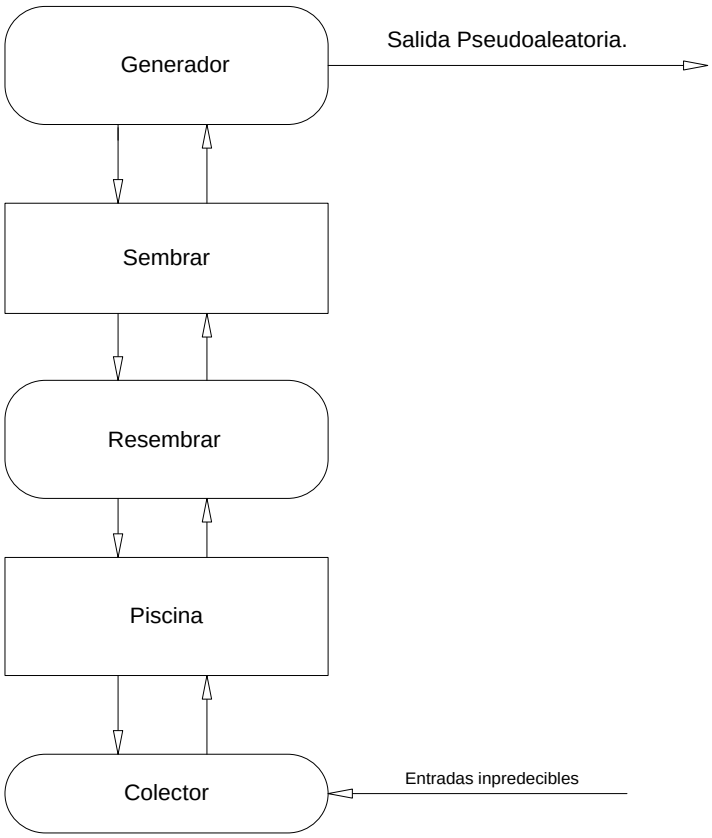
1. Un acumulador de entropía que colecciona muestras desde las fuentes de entropía y las almacena en dos piscinas.

2. Un mecanismo de resiembra el cual sirve periódicamente de semilla en la generación de la clave, con nueva entropía en las piscinas. El proceso de combinar la clave existente y las nuevas muestras dentro de una nueva clave , recibe el nombre de resemillar.

3. El mecanismo de generación que genera salidas de claves del PRNG.

4. El control de resiembra, el cual se determina cuando una resiembra se esta realizando.

Figura 5. Generalización de un PRNG con resembrado periódico




A continuación se especifica el rol de cada componente en el diseño del PRNG y los requerimientos para cada componente en términos tanto de seguridad como de desempeño. También se especifica la forma como cada componente puede interactuar con los otros.

**3.1.1 Acumulador de entropía (Colector).** A partir de las secuencias de entrada se llega a una colección con bastante entropía, la aplicación de la función de mezcla $h$ las transforma en secuencias concatenadas. Lo anterior se aplica a muestras de cada fuente a cada piscina en forma alternativa.

La acumulación de entropía es el proceso de inicialización por el cual el PRNG adquiere un nuevo estado interno no adivinable. Durante la resiembra es indispensable que la entropía se acumule completamente desde las muestras, con el fin de evitar los ataques iterativos de adivinación [6]. Es importante tambien que se estime correctamente la cantidad de entropía que se tiene que coleccionar y ésta tiene que ser muy grande. El mecanismo de acumulación de entropía puede también resistir el ataque de selección de entrada [6],

En Yarrow, la entropía de las muestras es coleccionada en dos piscinas, cada una en contexto de mezcla. Las dos piscinas son una piscina rápida y una piscina lenta. La piscina rápida proporciona resiembras frecuentes de la clave , con el propósito de asegurar en lo posible que la clave comprometida tenga una corta duración y la medicion de la precision de los estimativos de la entropía de cada fuente. La piscina lenta proporciona pocas y conservativas resiembras.

**3.1.2 Mecanismo de Re-Siembra.** El mecanismo de resiembra conecta al acumulador de entropía con el mecanismo de generación.

Esto es posible cuando el control de resembrado determina que esto se requiere, entonces el componente de resembrado modifica la clave y lo hace utilizando el mecanismo de generación, con información de una o ambas piscinas y que son actualizadas por el acumulador de entropía, de tal forma que el objetivo sea que la clave o la piscina sean desconocidas por el adversario, después del resembrado.

El resembrado de la piscina rápida usa las claves actualizadas y todas las entradas mezcladas para la piscina rápida ya que él último resembrado genera una nueva clave. Después de que esto se hace, la estimación de la entropía para la piscina rápida es inicializada a cero.

El resembrado de la piscina lenta utiliza las claves actualizadas y todas las entradas mezcladas para la piscina rápida y todas las entradas mezcladas para la piscina lenta para generar una nueva clave. Después de hecho esto, la estimación de la entropía para ambas piscinas es inicializada a cero.

El mecanismo de resiembra genera una nueva clave $K$ para el generador desde la piscina del acumulador de entropía y la clave existente. El tiempo de ejecución del mecanismo de resiembra depende de un parámetro $P_t > 0$. Este parámetro puede estar fijado para la implementación o ser dinámicamente ajustable.

El proceso de resiembra consta de las siguientes etapas:

1. El acumulador de entropía computa el hash sobre la concatenación de todas las entradas dentro de la piscina rápida, El resultado se llama $v_0$
2. Se asigna $v_i := h(v_{i-1} \mid v_0 \mid i) \, para \, i = 1,...,t.$
3. Se asigna $K \longleftarrow h'(h(v_{P_t} \mid K), k).$
4. Se asigna $C \leftarrow \mathrm{E}_k(0)$.
5. Inicializa a cero todos los acumuladores de estimativos de entropía en el acumulador de entropía.
6. Limpia la memoria de todos los valores intermedios.
7. Si un archivo de semilla está en uso, los próximos $2k$ bits de salida del generador están escritos para el archivo de semilla, entonces se sobre-escribe el valor presente.

En la etapa 1 se recolecta la salida del acumulador de entropía. En la etapa 2 se utiliza una fórmula iterativa de longitud $P_t$ y se hace la resiembra, que es computacionalmente costosa. En la etapa 3 se utiliza una función hash $h$ y una función $h'$, la cual crea una nueva clave $K$ a partir de la actualizada y un nuevo valor de entropía. La etapa 4 define un nuevo valor para el contador.

La función $h'$ esta definida en términos de $h$. Para computar $h'(m,k)$ se construye así:

$$s_o = m$$
$$s_i = h(s_0 \mid ... \mid s_{i-1})\ i = 1,...$$
$$h'(m,k) := \textit{first}\ \text{k bits of}\ (s_0 \mid s_1 \mid ...)$$

Esto es efectivamente una función conocida como "adaptador de tamaño" que convierte una entrada de cualquier longitud en una salida de longitud específica.

Si la entrada es más grande que la salida, la función toma el bit adicional de la entrada. Si la entrada es del mismo tamaño que la salida, la función es denominada de identidad. Si la entrada es más pequeña que la salida, se generan unos bits extras usando la función hash.

Si no hay seguridad se asigna un nuevo valor al contador C. Esto se hace para permitir más flexibilidad en la implementación y aun mantener compatibilidad entre las diferentes implementaciones.

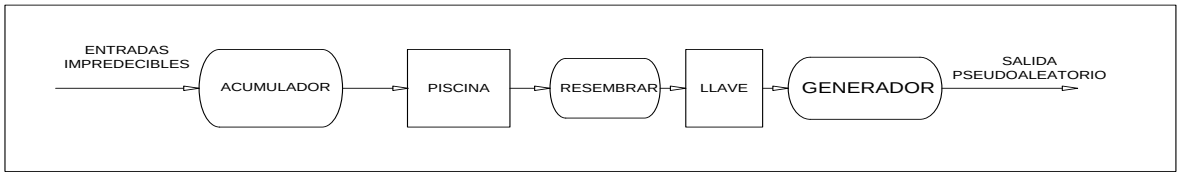
Asignándo el contador C, se hace simple para que una implementación genere completamente un búfer de salida única desde el generador. Si la resiembra ocurre, la nueva salida puede derivarse desde una nueva semilla y no de una vieja salida del búfer.

Asignándo un nuevo valor C se hace más simple, cualquier dato en la salida del búfer, se descarga simplemente. Se re-utiliza el valor del contador existente que no es compatible con las diferentes implementaciones que tienen diferentes tamaños en los búfer de salida y, así, el contador tiene avances a diferentes puntos. El rebobinado del contador en una posición virtual actualizada es considerado un error.

La resiembra en la piscina lenta alimenta el hashing de la piscina lenta dentro de la piscina rápida; entonces se presenta la resiembra. En general esta resiembra lenta puede tener un conjunto $P_t$ alto y tolerable.

**3.1.3 Mecanismo de generación.** Se tiene un valor de contador C de n-bits. Para generar los próximos bloques con n-bits de salida, se incrementa C y se cifra con el cifrador de bloque usando la clave K.

Figura 6. Diagrama de Yarrow generando bloques




Para generar el próximo bloque de salida, se ejecutan las siguientes instrucciones:

$$C \longleftarrow (C+1) \bmod 2^n$$

$$R \longleftarrow E_K(c)$$

Donde R es el próximo bloque de salida y $K$ es la clave actualizada del PRNG.

Sí la clave está comprometida en cierto punto en el tiempo, el PRNG no debe producir lentamente muchos rendimientos de salida anteriores a las que fueron generadas antes del compromiso.

Es claro que este mecanismo de generación no es inherentemente resistente a una clase de ataque. Por esta razón se investiga cuántos bloques tienen rendimiento. Una vez que se alcanza algún límite $P_g$ ( parámetro de un sistema de

seguridad $1 \le P_g \le 2^{n/3}$), se genera k bits de salida del PRNG y se usan estos como nueva clave.

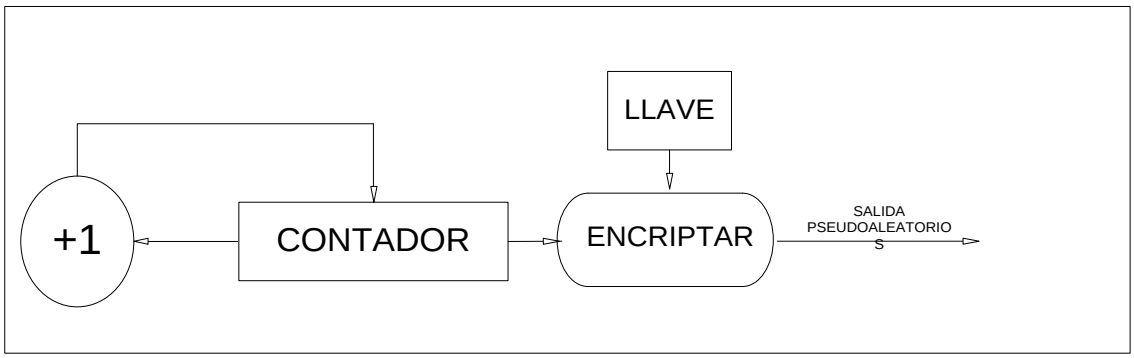
$$k \longleftarrow \textit{proximo}\ \text{k bits de la salida del PRGN}$$

Se llama esta operación un generador de puerta. Nótese que todo no es una operación de resiembra y ninguna nueva entropía se introduce dentro de la clave.

Con el interés de guardar un diseño conservativo extremo, el máximo número de salidas del generador entre resiembras está limitado a $min\left(2^n, 2^{k/3} P_g\right)$ bloques de salida de n-bits. El primer término en el mínimo previene el valor de C en el ciclo.

El segundo término hace que sea extremadamente improbable que K repita el mismo valor dos veces. En la práctica $P_g$ puede ser un conjunto más pequeño que éste, por ejemplo, en el orden de minimizar el número de salidas que pueden ser aprendidas por retro-alimentación.

Figura 7. Mecanismo de generación



**3.1.4 Control de resiembra.** El mecanismo de control de sembrado se presenta teniendo en cuenta las siguientes consideraciones. Frecuentemente el resembrado es deseable, pero probablemente es vulnerable a un ataque de adivinación iterativo [6]. Un resembrado poco frecuente, permite actuar al adversario, siempre que tenga comprometida la clave con más información.

Existe control en los estimativos de entropía para cada fuente, así, como es que llegan las muestras a cada piscina. Cuando cualquier fuente en la piscina rápida ha pasado el valor del umbral, entonces se resiembra desde la piscina rápida. En muchos sistemas, se puede esperar que esto pase muchas veces en una hora.

Cuando cualquier $K$ de las $n$ fuentes tiene un alto umbral en la piscina lenta, entonces se resiembra desde la piscina lenta. Este es un proceso mucho más lento.

Para Yarrow –160, el umbral para la piscina rápida es de 100 bits y para la piscina lenta es de 160 bits. Al menos dos diferentes fuentes están sobre 160 bits en la piscina lenta antes de la resiembra en la piscina lenta[6]. (Esto se facilita para diferentes ambientes, los ambientes con tres fuentes de entropía rápida son buenos y razonables).

Él módulo de control de resiembra determina cuándo la resiembra está siendo realizada. Ocurre cuando alguna aplicación explícita pregunta por la operación de resiembra. Esto se usa raramente y sólo por aplicaciones que generan valores muy altos de aleatoriedad. El acceso a funciones de resiembra explícita pueden ser restrictos en muchos casos.

Las Re-siembras[3] periódicas ocurren automáticamente. La piscina rápida se usa para la resiembra, sin embargo cualquiera de estas fuentes tiene una entropía estimada sobre algún valor de umbral. La piscina lenta es usada para resembrar, sin embargo, al menos dos de las fuentes tienen estimativos de entropía antes que otros valores de umbral.

## 3.4 ESTIMACIÓN DE LA ENTROPÍA

La estimación de la entropía es el proceso de determinar, cuánto trabajo puede tomar un adversario para adivinar el contenido actual de las piscinas.

El método general de Yarrow parte del grupo de muestras en fuentes y de los estimativos de las contribuciones de entropía de cada fuente separadamente. Para hacer esto se estima la entropía de cada muestra separadamente y entonces se adicionan estos estimativos para todas las muestras que vienen de la misma fuente.

La implementación será costosa por la determinación de sus fuentes. Las fuentes no están cerradamente encadenadas, o exhiben cualquier correlación significativa.

La entropía de cada muestra es medida en tres formas:

1. El programador suministra un estimativo de entropía en una muestra cuando escribe la rutina para coleccionar datos de la fuente. Así el programador puede enviar en una muestra, un estimativo de 20 bits de entropía.

[3]Entropía: magnitud que mide la información contenida en un flujo de datos, es decir, lo que nos aporta sobre un dato o hecho concreto [Wikipedia]

2. Por cada fuente se utiliza un estimador estadístico de la muestra. Esta prueba se hace para descubrir situaciones anormales debido a que la muestra tiene una entropía muy baja.

3. Para determinar la máxima amplitud muestral, ”densidad” se opera con la longitud de la muestra en bits y multiplicando por algún factor constante menor de uno, esto con el propósito de obtener un estimativo máximo de entropía en la muestra. En Yarrow-160 se usa un multiplicador de 0.5.

Se usa el valor más pequeño de estos tres estimativos como entropía de la muestra en cuestión.

La prueba estadística especificada en uso depende de la naturaleza de la fuente y puede ser cambiada en diferentes implementaciones. Esto es justamente otro componente, el cual puede ser sustituido y remplazado por los componentes bien preparados en diferentes ambientes.

## 3.5 GENERANDO SALIDAS SEUDO-ALEATORIAS

El mecanismo de generación proporciona la salida del PRNG. La salida tiene que ser apropiada, si un adversario no conoce la clave del PRNG, él no puede distinguir las salidas del PRNG de una verdadera secuencia aleatoria.

El mecanismo de generación puede tener las siguientes propiedades:

1. Resistencia al ataque cripto analítico.
2. Eficiente.
3. Resistente a la retroalimentación después de una clave comprometida
4. Capacidad de generar secuencias muy largas de rendimiento firme sin resembrar.

Hay dos puntos del plan principal que deben ser tenidos en cuenta con este generador seudo-aleatorio. El primero tiene en cuenta tanto la criptografía, como la calidad de sus entradas por su seguridad y calidad de los datos. El segundo consiste en que el mecanismo del rendimiento y piscinas de entropía son tan distintos como es posible.

A partir de un ejemplo con dimensiones muy pequeñas, se explica el funcionamiento del cifrador basado en una permutación seudo-aleatoria.

## 3.4 FUNCIONAMIENTO DEL CRIPTOSISTEMA

**3.4.1 Proceso para Generar las claves a partir del PRNG.** A continuación se explicaran las aplicaciones que tienen que ser compiladas para implementar el esquema propuesto, estas son:

- entropyhooks - colección de rutinas de entropía . Contiene DLL para coleccionar entropía, proporciona rutinas de enlace para coleccionar los eventos del temporizador del ratón y teclado, así como las rutinas de comunicacion basadas en el mapeo de la memoria y archivos y banderas para los eventos.
- prngcore - Las rutinas principales del PRNG Actualmente en forma DLL, con algunas funciones disponible para un usuario final, y otras disponible sólo para un usuario a nivel alto.
- frontend - Programa de colección de entropía e instalación del prng. Esta aplicación tiene que correrse para cronometrar la inicialización en cualquier sistema que desee usar las rutinas del prng. Si se Intenta llamar las rutinas del prng sin correr frontend se llegara a un error.
- smf – Secure Malloc/Free. Contiene DLL para la administración de la memoria. La memoria es asignado desde el proceso del sistema que

compagina el archivo y el mapeo del espacio de dirección. La memoria se anulaseguramente al liberarse.

- zlib – Compila versiones de la biblioteca de compressiong de zlib. zlibr.lib es una versión release y zlibd.lib es versión debug.
- Cifrar –Compila y ejecuta el proceso de encriptar un archivo de texto.
- Descifrar –compila y ejecuta el proceso de desencriptar un archivo cifrado.

**3.4.2 Rutinas del PRNG en el criptosistema.** En el programa de encriptación se hace uso de las siguientes rutinas fundamentales del generador de números seudo-aleatorios:

Nota: todos se deben especializar en rutinas que retornan eventos o errores cuando se valoran para su funcionamiento.

- int prngOutput(outbuf,outbuflen)

Escribe el valor de outbuflen de datos "aleatorios" desde outbuf. Esta rutina tiene retroalimentación de protección, pero se debe llamar la rutina prngAllowReseed siempre que se pueda ahorrar los ciclos para garantizar buen rendimiento.

- int prngStretch(inbuf,inbuflen,outbuf,outbuflen)

Toma los bytes de inbuflen a partir de los datos de inbuf y los revuelve en los bytes de datos de outbuflen almacenándolos en outbuf.

- prngInput(inbuf,inbuflen,poolnum,estbits)

Toma inbuflen bytes de datos de inbuf y los coloca en la piscina de entropía poolnum. El nombre de la piscina de entropía del usuario puede encontrarse en los nombres en la librería usersources.h

- int prngForceReseed(ticks)

Fuerza una resemilla sobre los últimos tick de longitud ticks. Se debe tener mucho cuidado al usar esta función para asegurar que no se produzca un estado de rendimiento pobre. Se sugiere que se use la función prngAllowReseed a cambio.

- Int prngAllowReseed(ticks)

Fuerza una resemilla, si hay bastante entropía. Un resemilla (de ticks longitud) se hace si el total de entropía estimada, ignora K fuentes grandes y es mayor que el UMBRAL. Actualmente, K = 0 (una mala idea) y el UMBRAL = 100, probablemente para asegurar que estos valores puedan llegar a producirse y revisarse, en userdefines.h retorna PRNG_ERR_NOT_ENOUGH_ENTROPY si no hay bastante entropía en la piscina en este momento.

- Int prngProcessSeedBuffer(buf,ticks)

Toma 20 bytes de datos de buf y los revuelve dentro de la piscina de entropía y entonces forza una resemilla de longitud ticks. Los primeros 20 bytes de rendimiento son retornados en buf, para el uso futuro de esta función. Se recomienda que los datos usados con esta función se almacenen en forma muy segura.

- Int prngSlowPoll(pollsize)

Hace una votación lenta para coleccionar una cantidad grande de datos vagamente aleatorios del propio OS. La votación con colecciones de muchos pollsize bytes puede usarse para controlar (aproximadamente) la longitud de la votación. Los datos reunidos se alimentan en la piscina de entropía. Si es su

decisión, se puede después de llamar esta función llamar la funcion que forza una resemilla,

**3.4.3 Ejemplo que ilustra el proceso de cifrado y descifrado.** El algoritmo criptografico es un sistema simétrico de cifrado por bloques, por tanto utiliza la mismas claves $(K_1, K_2)$ para el proceso de cifrado como para el proceso de descifrado, su diseño permite la utilización de claves de sistemas de longitud variable siempre que sea múltiplo de 8 bits. La longitud de las claves utilizadas por defecto es de 120 bits. De la misma manera el algoritmo permite la utilización de bloques de información con un tamaño variable siempre que sea múltiplo de 8 bits, siendo el tamaño mínimo recomendado de 120 bits (El tamaño mínimo de 15 bytes). Este algoritmo funciona a nivel de bits. Teniendo en cuenta que la funcion de permutación es sencilla y con distribución uniforme; aunque matemáticamente es trivial, requiere operar computacionalmente a bajo nivel.

En el proximo párrafo se quiere realizar un ejemplo hipotético que demuestra como funcionan los procesos de cifrar y descifrar bloques del archivo de texto o del texto cifrado respectivamente. Esto es simplemente la abstracción del esquema propuesto por Evan y Mansour [1] que es la base de la implementacion pero enriquecido con la funcionalidad PRNG para generar las claves.

**3.4.3.1 Proceso de cifrado.** Se tiene en cuenta la expresión matemática que formaliza este proceso $C = E_K(M) = F(M \oplus K_1) \oplus K_2$ y teniendo en cuenta lo anterior se establecen tamaños hipotéticos para la clave y el bloque y se generan secuencias de dicho tamaño para las posibles claves o sea $k_1$ y $k_2$.

Se considera un tamaño de bloque por ejemplo de 8 bits sea $b_1 = 10110111$ correspondiente al bloque texto y el PRNG genera una clave también de 8 bits por ejemplo $k_1 = 11101011$.

Sea $b_1 \oplus k_1 = 10110111 \oplus 11101011 = 01011100$. A partir de esta operación se seleccionan en forma aleatoria dos números diferentes, en este caso entre 1 y 8 por ejemplo que sean 2 y 7, se intercambian los bits de las posiciones 2 y 7, de la secuencia $01011100$ Esto permite producir una de sus permutaciones, que en este caso es $P_1 = 00011110$ . Ahora, utilizando PRNG, generamos una nueva clave, por ejemplo, $k_2 = 10101010$ .Sea $p_1 \oplus k_2 = 00011110 \oplus 10101010 = 10110100$ que corresponde al bloque cifrado. La clave compuesta para el proceso de descifrar es $k_1 = 11101011$ y $k_2 = 10101010$ y los indices de la permutación, buscando como medio el uso de un canal seguro.

**3.4.3.2 Proceso descifrado.** La descripción del proceso de descifrado del algoritmo criptografico es sencilla. Consiste en sustituir las transformaciones utilizadas en el cifrado por sus inversas e invertir el orden de aplicación de dichas transformaciones o funciones matemáticas.

Teniendo en cuenta esto el proceso de descifrado es descrita, por la siguiente ecuación: $M = D_K(C) = F^{-1}(C \oplus K_2) \oplus K_1$

El bloque cifrado es $Q_1 = 10110100$ ahora con la clave del proceso anterior $k_2 = 10101010$ se opera $Q_1 \oplus k_2 = 10110100 \oplus 10101010 = 00011110$ , ahora se opera con la permutación inversa intercambiando los bits de la posición 2 y 7 de la secuencia $00011110$ se produce $p_2 = 01011100$ , ahora operando en el ámbito de bit se llega a $p_2 \oplus k_1 = 01011100 \oplus 11101011 = 10110111$ .

Se puede observar que se llega a la misma secuencia con la que se inicio el proceso de cifrar el bloque del archivo de texto.

El proceso experimental se realiza con bloques y claves de 120 bits.

**3.4.4 Confidencialidad.** Este Item es una posible respuesta al interrogante ¿Cómo acuerdan los usuarios las claves para cifrar y descifrar ? El sistema propuesto esta dentro la criptografia simetrica , que es sistema de cifrado más antiguo y consiste en que tanto el emisor como el receptor encriptan y desencriptan la información con una misma clave $K(K_1, K_2)$ (clave secreta) que ambos comparten. El funcionamiento es muy sencillo: el emisor cifra el mensaje con la clave $K(K_1, K_2)$ y se lo envía al receptor. Este último, que conoce dicha clave, la utiliza para desencriptar la información.

Es importante considerar que para que el sistema sea razonablemente robusto contra ataques de tipo criptoanálisis, esta clave $K(K_1, K_2)$ ha de ser mayor de 40 bits, lo cual choca con las restricciones de exportación de tecnología criptográfica del gobierno americano, que marca los 40 bits como límite de clave para programas que utilicen este tipo de tecnología.

Algoritmos típicos que utilizan cifrado simétrico son DES, IDEA, RC5, etc. El criptosistema de clave secreta más utilizado es el **Data Encryption Standard(DES)** desarrollado por IBM y adoptado por las oficinas gubernamentales estadounidenses para protección de datos desde 1977.

Este sistema de cifrado tiene la **ventaja** de que es altamente eficiente, dado que los algoritmos utilizados son muy rápidos al poder implementarse tanto en hardware como en software de una forma fácil.

El mayor **inconveniente** de la criptografía simétrica es que esta clave *k*, al ser compartida, ha de ser comunicada de forma segura entre las dos partes de la comunicación (por teléfono, correo certificado, etc.), previamente a ésta. Si este secreto fuese enviado por un canal inseguro, como por ejemplo Internet , la seguridad del sistema sería bastante pobre, dado que cualquiera podría

interceptarla y comprometer todo el sistema. También hay que tener en cuenta la frecuencia con la que esta clave debe ser renovada para evitar que sea desvelada.

Ambos problemas son resueltos con la llegada de los criptosistemas de clave pública o asimétrica.

Figura 8. Diagrama de flujo sobre el proceso de encriptación.

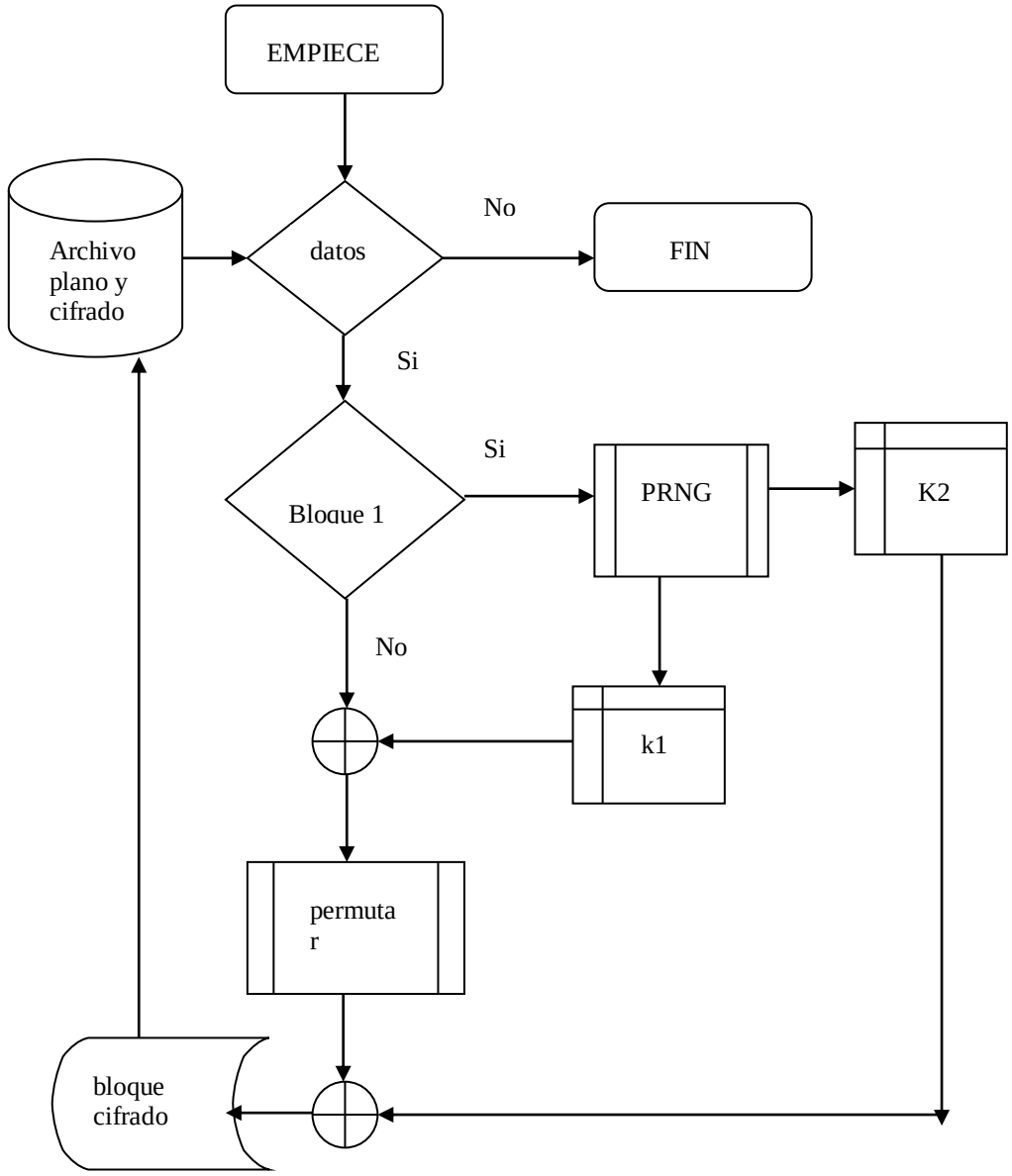
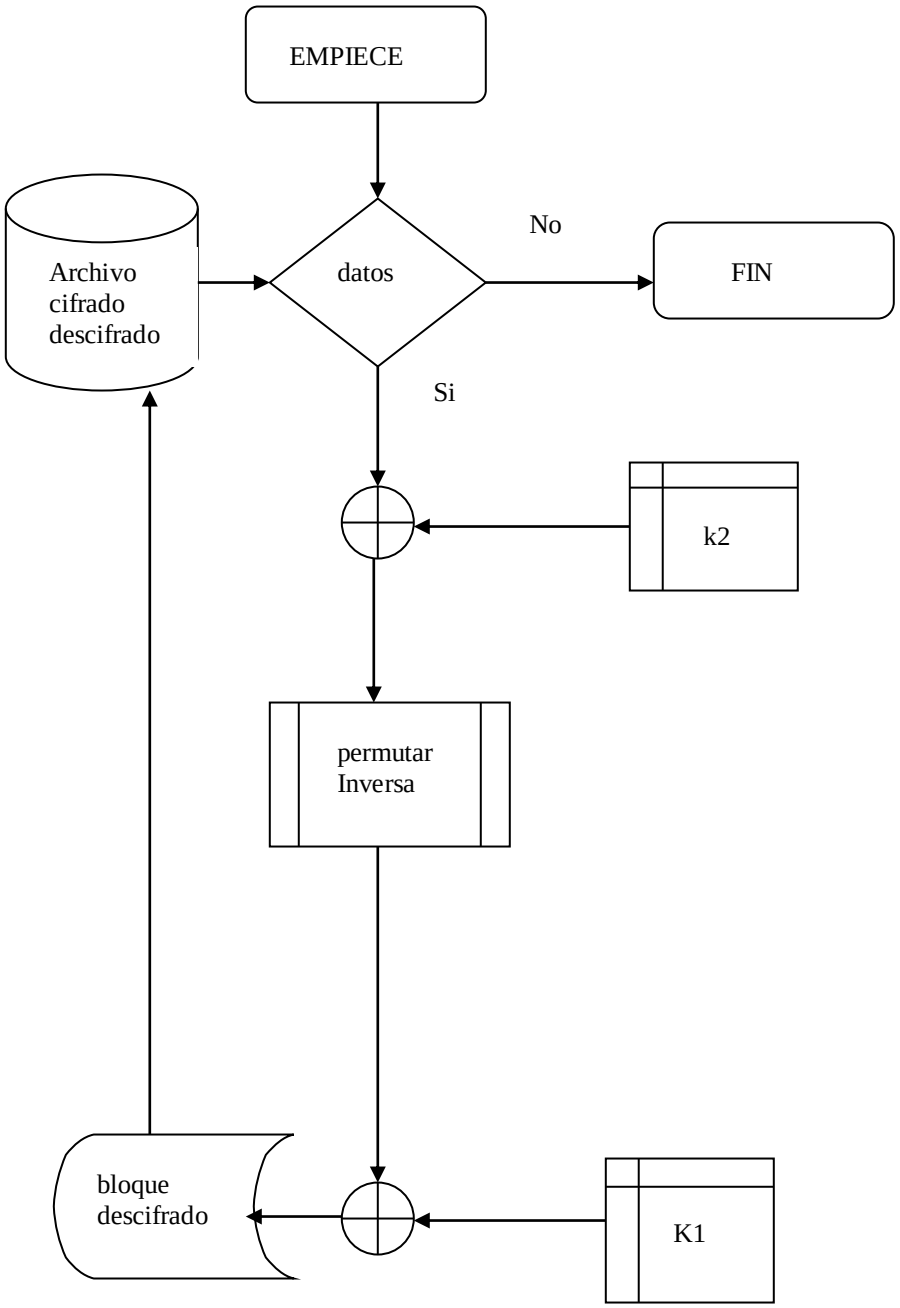


Figura 9. Diagrama de flujo, proceso de desencriptación.



# 4. RESULTADOS

## 4.1 RESULTADOS OBTENIDOS

El proceso investigativo del presente trabajo de grado se inicia con el conocimiento del esquema propuesto por Even y Mansour, y fue necesario entender qué es un cifrador de bloque y cómo una sola permutación es seleccionada en forma aleatoria y utilizada, tanto en el proceso de cifrado como en el descifrado. Con base en los resultados, se puede ver inmediatamente, que la clave que modifica la permutación es muy sencilla y fácil de implementar.

A partir de la anterior aserción se mostrará, como entrada, un archivo de texto; por ejemplo, un programa fuente de Karel con muy pocas líneas.

```
// File choreographer.kpp  (from Karel++ textbook
//  by Bergin,et.al. see p. 56)
// Run this on file "harvest.wld"

class Choreographer : ur_Robot
{
    ur_Robot Lisa(4,2, East, 0);  // the 1st helper robot
    ur_Robot Tony(6,2, East, 0);  // the wnd helper robot

    void harvest();
    void harvestARow();
    void harvestCorner();
    void move();
    void pickBeeper();
    void turnLeft();
    void turnOff();
};

void Choreographer::harvest()
{
    harvestARow();
    turnLeft();
    move();
    turnLeft();
    harvestARow();
}
```

```
task
{   Choreographer Karel (2,2, East, 0);
    Karel.harvest();
    Karel.turnOff();
}
```

Es necesario activar el código de la aplicación FrontEnd ( también llamado colector de entropía). Este programa puede ejecutarse al iniciar y ,también , antes de que el cliente llame los procedimientos públicos de la aplicación prngcore.

Este programa conecta las DLL de enlace y las del PRNG, las entropías de datos son enviadas unas tras otras, y luego dentro del pool de entropía, el cual ejerce el control. El solo control que el usuario tiene sobre el contexto del menú es el que se hace en el área del icono.

Dentro del procedimiento de aplicación , se realiza a continuación la ejecución del programa que cifrara el archivo de entrada.

L"š &Xü_f–ÅiV { ÒO'*F›™tÆ• ; n
’š .Fü¥.÷□oV h Ž"$B>—'$ü<b <y Þ !cQ½"d˜€;@ yCêãoz ˜ø
¬’;a
rCá"&< ûà$ä_wV^>
ÒZ9*G½"s˜_9>t iÔ_.<G”çl• >~\ n ê"*= • _p ôI\ s Ý”$B>”_$üoil,s Ž^o ]¾š,õ
• )^Y □^co ˜Å$ü’4
t <Ö,; ó• hÆOi
s Ž^BE ”_$‘>Da ~
áš
Zî 2¨ê7 ;} áÑo□ ¡_$¬’;G yCš'+o\ü¥t”>;A
~á¤Eo ”_ T• ; ^j
¶~o'UÏÓ`…_3 E i<šooBûâ`ü_zA y á« Cç"?zÆ; ^< Ž +o\úØr”sop n

×Zgf êú$ü• ;E u <'
9Qç">zÆ; ^< Ž +oD¨èn¥O~C nK'€BE ”_$†ÅrW^h ë' *R½ ,ù„ ^<Cí"&+
½ÔvìÝ}UV5XÝ""t9šøÅrW^_
ŽZ* SÏšt–Oi
Dt ëS*<@ç" T' 9^<C<".=Bü×pã¯tDV5XÝ”oo ”• p ÎWV hK'€BE ”_$çÅm
VV5XÝ”oo ”• p ÎWV hK'€BE ”_$–^iE o ˆ©
8 ˜Å T" 9^<C<¤Eo ”_ T_z@ iÑšoowóãv”Å|AI
×Zo UÏ• hüµ)L0CŒ_<; ”_-ù„ ^<C…_=*X›åe Y~@
4J_¤Eo ”_N•>~_Ph ë' )Rç">zÆf>t I

Para comprobar la terminación de este trabajo de grado, también se implementó el programa que permite descifrar el archivo presentado anteriormente, produciendo el siguiente resultado.

```
// File choreographer.kPp  (from Karel++ textbook
//  by Bergin,et.Al. see p. 56(
// Run this on file "harvest.Wld"

class CHoreographer : Ur_Robot
{
    ur_Robot Lisa(4,2, East, 0);  // the 1st heLper robot
    ur_Robot Tony(6,2, East, 0);  // the wnd helper robot

    void harvEst();
    void harvestARow()
    void harvestCorner();
    void move()
    void pickBeeper();
    void turnLeft(      ;
    void tuRnOff();
};

void ChoreograPher::harvest()
{
    harvestARow();
    tUrnLeft();
    move();
    tUrnLeft();
    harvestARow((;
}
```

```
task
{   ChoReographer KareL (2,2, East, 0    ;
    Karel.hArvest();
    Karel.turnOff()
}
```

Se puede observar que los archivos presentan características similares para un formato dado, pero no podemos afirmar que son dos copias exactamente iguales para todo tipo de formato de edición.

Finalmente, para demostrar la hipótesis sobre el uso de una sola permutación en el esquema propuesto y el uso de un generador de números seudo-aleatorios se realizó un programa que implementa cinco pruebas estadísticas que son utilizadas generalmente para determinar si una secuencia binaria, como son las permutaciones generadas en esta implementación posee características específicas de secuencias seudo-aleatorias verdaderas, como probablemente se ha declarado.

Los resultados para la primera permutación en el ejemplo anterior son los siguiente:

```
Secuencia 1

01010110001110111000000111111100010001010001100101111000010101010101011001
10011000011111101001000100111111000010110101001
Test de Frecuencia
Numero de 1 's =59
Numero de 0 's =61
estadistico chi cuadrado 0.000
  Test Serial
Numero de 0´s y 0's = 30
Numero de 0's y 1's=31
Numero de 1 's y 0's =31
Numero de 1 's y 1's = 28
estadistico chi cuadrado 1.000
 Test Poker
Numero de 0´s y 0's y 0's = 15
Numero de 0´s y 0's y 1's = 15
Numero de 0´s y 1's y 0's = 20
Numero de 0´s y 1's y 1's = 11
Numero de 1´s y 0's y 0's = 15
```

```
Numero de 1´s y 0's y 1's = 15
Numero de 1´s y 1's y 0's = 11
Numero de 1´s y 1's y 1's = 17
estadistico chi cuadrado 326.000
 Run Test
Numero de gaps de longitud 1 = 15
Numero de bloques de longitud 1 = 20
Numero de gaps de longitud 2 = 15
Numero de bloques de longitud 2 = 11
estadistico chi cuadrado 27.000
Test de correlacion
Numero de bits no iguales a sus d=3 mas delante = 65
estadistico chi cuadrado 148.822
```

Estos valores para el estadístico $\chi^2$ reciben el nombre de valores calculados ; y comparándolos contra los valores criticos de $\chi^2$ en la tabla 5.1 y 5.2 en [5] para secuencias de tamaño n=120 , nos permitirá comprobar o rechazar la hipótesis de esta investigación.

En la tabla a continuación para cada prueba estadística muestra la formula del estadístico $\chi^2$ con los respectivos grados de libertad y los valores calculados con el programa de test estadístico [ver apéndice D] y los respectivos valores criticos con un nivel de significancia $\alpha = 0.05$.

La tabla nos permite establecer que para la secuencia mostrada anteriormente las pruebas de frecuencia y la prueba serial y se quedan las pruebas poker , test y correlacion.

| Prueba estadística | Formula para chi-cuadrado | G.L | $\chi^2$ calculada | $\chi^2$ Estimada |
|---|---|---|---|---|
| Frecuencia | $X_1 = \frac{(n_0 - n_1)^2}{n}$ <br> $n_0 = 59$ y $n_1 = 61$ | 1 | 0.033 | 3.8415 |
| Serial | $X_2 = \frac{4}{n-1}\left(n_{00}{}^2 + n_{01}{}^2 + n_{10}{}^2 + n_{11}{}^2\right)$ <br> $-\frac{2}{n-1}\left(n_0{}^2 + n_1{}^2\right) - 1$ | 2 | 1 | 5.9915 |
| Poker | $X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i{}^2\right) - k$ | $2^m - 2$ =6 | 326 | 12.5916 |
| Run | $X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} +$ <br> $\sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$ | $2k - 2$ =6 | 27 | 12.5916 |
| Correlacion | $X_5 = 2\frac{\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$ | $N(0,1)$ | 148.822 | 1.96 |

# 5. CONCLUSIONES

A lo largo de este trabajo se ha profundizado en el estudio de la estructura interna de un criptosistema en el orden de la construcción de un cifrador basado en una sola permutación y su respectivo PRNG, un generador de secuencias seudo-aleatorias, para generar las respectivas claves. Con las dificultades que obedecen al análisis y definicion de un conjunto de los posibles ataques realizados por los adversarios.

Even y Mansour [1] establecieron que la construcción de un cifrador mediante una permutación no requería la necesidad de almacenar o generar una multitud de permutaciones y con esta construcción evidentemente no se requirió almacenamiento de permutaciones que confirma dicha afirmación.

Muchas opiniones corroboran que el diseño y análisis de algoritmos convencionales están basados en las propiedades de difusión y confusión. Desde el punto de vista de permutaciones no seleccionadas éstas se construyen usando primitivas realizables. Para la seguridad de los algoritmos, es mejor evaluarlos a la luz de los métodos cripto-analíticos y los principios existentes.

La afirmación de que los PRNG son de diferentes clases de primitivas criptográficas, hay aplicaciones donde el desempeño de PRNG es un problema serio que amerita un nuevo algoritmo. Yarrow-160 en la práctica tiene debilidades en la estimación de la entropía y no en el criptoanálisis. Es necesario continuar probando los mecanismos de la entropía y ésta estará sujeta a futuras investigaciones.

Las reglas de control de re-sembrado es aún un diseño ad-hoc; el estudio extenso podría rendir una mejora en el control de reglas en el proceso re-sembrar.

No está establecido cómo se resiste el compromiso de estado del sistema propuesto. Esto constituye un enorme problema práctico que ha recibido poca atención en la literatura.

**Aportaciones prácticas.**

- Se analizó el código para las DLL que contiene las funciones necesarias para coleccionar y guardar datos de los eventos temporizadores del ratón y del teclado. Los datos están escritos mediante archivos de trazado de memoria controlados por este par de eventos. Un problema mayor en la programación de las DLL es que hace llamadas a una variedad de diferentes procesos de medios que maneja y no pueden guardar las situaciones de memoria como tal.

- Se estudió la información que describe las estructuras de los datos usadas en el conjunto de operaciones prngcore, así como las rutinas de nivel alto para su utilización, por aquéllos usuarios que estarán preparados para alterar el prng e inicializar la conexión entre los prng y las rutinas de colección de entropía.

- Se estudió el código para la DLL que contiene todas las funciones necesarias para la asignación de memoria en forma segura y su eliminación. Esta biblioteca es nueva y requerirá alguna comprobación seria. Está teóricamente seguro y basado en Gutmann 1996, el papel en el tema (vea Usenix 1996).

- Fue estudiado y probado el código para el FrontEnd (Es llamada para la Colección de Entropía) y su respectiva aplicación. Este programa debe correrse al iniciar, y debe estar corriendo ante cualquier cliente que llama las DLL que pueden hacerse con el programa prngcore.

- Se analizó y desarrolló el código para encriptar basado en el esquema propuesto, una aplicación muy sencilla que demuestra el arreglo mínimo y exigió conectarse una vez a la piscina de entropía la cual ha sido fijada por el proceso encriptar. Las funciones principales son llamadas por el OS. Esta función apenas llama cada uno de las rutinas del prng necesarias para crear las claves utilizando el programa de almacenamiento de clave.

- Se analiza y desarrolla un programa sencillo para desencriptar utilizando el programa que almacena las claves y cuya copia debe estar en propiedad del usuario que descifra.

## BIBLIOGRAFIA

[1]. S. Even and Y. Mansour, A Construction of a Cipher from a Single Pseudorandom
Permutation, Journal of cryptology,Volume 10, Number 3, 1997, Pages 151-162

[2] C.E. SHANNON, Comunication theory of secresy system , Bell system Teech. J, Vol. 28,1949,pp, 656-715.

[3] O. GOLDREICH, S Goldwaser and S. Micali , “How To Construct Random Function “ Proceeding of the –25th Annual Symposium of Foundation of computer science, 24-26, 1984.

[4] J. DAEMEN, “Limitation of the even-Mansour Construction “ , Katholieke universiteit Leuven Laboratorium Esat , B-3001 heverloe Belgium.

[5] A. MENEZES, P. Van Oorchot , Scott a: castone , “ Hand Book Applied Cryptography “ , CRC press , 1997

[6] J. KELSEY, B. Schnneir , D. Wagner and C. Hall , “Cryptanalytic Attack on Pseudorandom Number Generators “ , fast software Encryption , 5th international Workshop Proceedings Springer-Verlag , 1998 , pp 168-188.

[7] J. KELSEY, B. SCHNNEIR, N FERGUSON, “ Yarrow –160 Notes on the design an analysis of the Yarrow Criptographic Pseudorandom Number generator“, Counterpane System , 101 E Minnehaha Parkway , Minneapolis , MN 55419, USA.

[8] J. VILLALOBOS, “Tipo abstracto de datos en lenguaje c “, Mc Graw Hill , 1995

[9] M. LUBY and C. RACKOFF, “How to construct pseudorandom permutation from pseudorandom function , SIAM Journal computing , 373-386, april 1988.

[10] N. HOLLAND“ Stream Cipher and Number Theory “ ,North_holland mathematical Library , volume 55 , Elsevier , 1998

[11] SHIMON Even, MANSOUR , Yishay: A Construction of a Cipher From a Single Pseudorandom Permutation, in ASIACRYPT 1991, pages 210-224

## APÉNDICE (S)

## PROGRAMAS DESARROLLADOS

A continuación se relacionan los programas específicos de la tesis para construir un cifrador basado en una permutación y con la utilización del PRNG (Yarrow-160) para generar las claves.

### Anexo A. Programas que implementa el TAD Lista que se utilizan para calcular la función inversa de permutación

```
/*-----------------------------------------------------------------*/
/*Conjunto de operaciones de Lista ,utilizada para la función inversa*/
/*-----------------------------------------------------------------*/
/* TAD Lista - Implementación #1 (Doblemente encadenada)          */
/*-----------------------------------------------------------------*/

#include <stdio.h>
/* #include <alloc.h> */

typedef unsigned char bytes;
typedef int TipoL;
/* #include "Lista1.h"*/

/*-----------------------------------------------------------------*/
/* Constructora: Crea una lista vacía. La ventana es indefinida    */
/* Complejidad:    O(1)                                            */

typedef struct ListaNodo
```

```
{   TipoL info1,info2;
    byte vec1[15] ,vec2[15];
    struct ListaNodo *ant, *sig;
}   *pListaNodo;

typedef struct
{   pListaNodo primero, ultimo, ventana;
    int longitud;
}   TLista, *Lista;

typedef pListaNodo Ventana;

Lista inicLista(void)
{   Lista resp;
    resp=(Lista)malloc(sizeof(TLista));
    resp->primero=resp->ultimo=resp->ventana=NULL;
    resp->longitud=0;
    return resp;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Coloca el nuevo elemento en la posición siguiente a */
/*              la ventana. Si la lista es vacía lo coloca de      */
/*              primero en la lista                                */
/* Complejidad:  O(1)                                              */

void anxLista(Lista lst, TipoL elem1,TipoL elem2,bytes *vecx,bytes  *vecy)
{   pListaNodo nuevo=(pListaNodo)malloc(sizeof(struct ListaNodo));
    int i,j;
        FILE *ptrtem;
```

```
    nuevo->info1=elem1;
    nuevo->info2=elem2;
          for (i=0;i<15;i++)
          nuevo->vec1[i]=vecx[i];


    for (i=0;i<15;i++)
          nuevo->vec2[i]=vecy[i];

/* temporal */
          ptrtem=fopen("c:\\tempctra.txt","w");

          for (j=0;j!=15; j++)
          {
       fprintf(ptrtem,"%d\n",vecx[j]);
          }
          for (j=0;j!=15; j++)
          {
       fprintf(ptrtem,"%d\n",vecy[j]);
          }
     fclose(ptrtem);
/* fin */


    nuevo->ant=nuevo->sig=NULL;
    if (lst->longitud==0)
       lst->primero=lst->ultimo=nuevo;
     else if (lst->ventana==lst->ultimo)
     { lst->ventana->sig=lst->ultimo=nuevo;
       nuevo->ant=lst->ventana;
```

```
    }
    else
    { nuevo->ant=lst->ventana;
      nuevo->sig=lst->ventana->sig;
      lst->ventana->sig->ant=nuevo;
      lst->ventana->sig=nuevo;
    }
   lst->ventana=nuevo;
   lst->longitud++;
}

/*-------------------------------------------------------------------*/
/* Modificadora: Coloca el nuevo elemento en la posición anterior a  */
/*               la ventana. Si la lista es vacía lo coloca de       */
/*               primero en la lista                                 */
/* Complejidad:  O(1)                                                */

void insLista(Lista lst, TipoL elem1,TipoL elem2)
{   pListaNodo nuevo=(pListaNodo)malloc(sizeof(struct ListaNodo));
   nuevo->info1=elem1;
   nuevo->info2=elem2;
   nuevo->ant=nuevo->sig=NULL;
   if (lst->longitud==0)
     lst->primero=lst->ultimo=nuevo;
    else if (lst->ventana==lst->primero)
    { lst->primero=lst->ventana->ant=nuevo;
      nuevo->sig=lst->ventana;
    }
    else
    { nuevo->sig=lst->ventana;
```

```
      nuevo->ant=lst->ventana->ant;
      lst->ventana->ant->sig=nuevo;
      lst->ventana->ant=nuevo;
    }
    lst->ventana=nuevo;
    lst->longitud++;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Elimina el elemento que se encuentra en la ventana  */
/*              de la lista. La ventana queda sobre el siguiente    */
/*              elemento. Si era el ultimo, la ventana queda        */
/*              indefinida.                                         */
/* Complejidad:  O(1)                                               */

void elimLista(Lista lst)
{   pListaNodo aux;
   if (lst->ventana==lst->primero)
   {  if (lst->ultimo==lst->primero)
           lst->ultimo=NULL;
      lst->primero=lst->primero->sig;
      free(lst->ventana);
      lst->ventana=lst->primero;
    }
     else
     { if (lst->ultimo==lst->ventana)
           lst->ultimo=lst->ultimo->ant;
        lst->ventana->ant->sig=lst->ventana->sig;
        if ( lst->ventana->sig!=NULL)
           lst->ventana->sig->ant=lst->ventana->ant;
```

```
        aux=lst->ventana;
        lst->ventana=lst->ventana->sig;
        free(aux);
    }
    lst->longitud--;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Mueve una posicion a la derecha la ventana. Si esta */
/*             sobre el ultimo elemento la ventana queda          */
/*             indefinida                                         */
/* Complejidad:  O(1)                                             */

void sigLista(Lista lst)
{   lst->ventana=lst->ventana->sig;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Coloca la ventana sobre el primer elemento       */
/* Complejidad:  O(1)                                             */

void primLista(Lista lst)
{   lst->ventana=lst->primero;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Coloca la ventana sobre el ultimo elemento       */
/* Complejidad:  O(1)                                             */

void ultLista(Lista lst)
```

```
{   lst->ventana=lst->ultimo;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Coloca la ventana sobre una posicion dada         */
/* Complejidad:  O(N)                                              */

void posLista(Lista lst, int pos)
{   int i;
    for ( lst->ventana=lst->primero,i=1; i<pos; i++)
          lst->ventana=lst->ventana->sig;
}

/*-----------------------------------------------------------------*/
/* Modificadora: Coloca la ventana en vent                         */
/* Complejidad:       O(1)                                         */

void situarLista(Lista lst, Ventana vent)
{ lst->ventana=vent;
}

/*-----------------------------------------------------------------*/
/* Analizadora: Retorna el elemento que se encuentra en la ventana */
/* Complejidad: O(1)                                               */

 pListaNodo infoLista(Lista lst)
{   return lst->ventana;
}

/*-----------------------------------------------------------------*/
```

```
/* Analizadora: Retorna la longitud de la lista                    */
/* Complejidad: O(1)                                               */

int longLista(Lista lst)
{   return lst->longitud;
}

/*-----------------------------------------------------------------*/
/* Analizadora: Indica si la ventana se encuentra indefinida       */
/* Complejidad: O(1)                                               */

int finLista(Lista lst)
{   return lst->ventana==NULL;
}

/*-----------------------------------------------------------------*/
/* Analizadora: Retorna la ventana                                 */
/* Complejidad:       O(1)                                         */

Ventana ventanaLista(Lista lst)
{ return lst->ventana;
}

/*-----------------------------------------------------------------*/
/* Destructora: Retorna la memoria ocupada por una lista           */
/* Complejidad:       O(N)                                         */

void destruirLista(Lista lst)
{ pListaNodo p,q;
  for(p=lst->primero;p!=NULL;)
```

```
  { q=p;
    p=p->sig;
    free(q);
  }
 free(lst);
}

/*-------------------------------------------------------------------*/
/* Persistencia: Carga una lista de un archivo                       */
/* Complejidad:       O(N)                                           */

Lista cargarLista(FILE *fp)
{ int longit,infox,infoy,i;
  bytes vecx[15],vecy[15];
  Lista lst=inicLista();
  fscanf(fp,"%d",&longit);
  for(;longit>0;longit--){
  { fscanf(fp,"%d",&infox);
    fscanf(fp,"%d",&infoy);
    for (i=0;i!=15;i++)
          fscanf(fp,"%d",&vecx[i]);
    for (i=0;i!=15;i++)
          fscanf(fp,"%d",&vecy[i]);
  }
    anxLista(lst,infox,infoy,vecx,vecy);
  }
  return lst;
}

/*-------------------------------------------------------------------*/
```

```
/* Persistencia: Salva una lista en un archivo                    */
/* Complejidad:       O(N)                                        */

void salvarLista(Lista lst, FILE *fp)
{
        int i;

        fprintf(fp,"%d\n",longLista(lst));
 for(primLista(lst);!finLista(lst);sigLista(lst)){

    fprintf(fp,"%d",lst->ventana->info1);
        fprintf(fp,"%d",lst->ventana->info2);
        for (i=0;i!=15;i++)
        fprintf(fp,"%d",lst->ventana->vec1[i]);
        for (i=0;i!=15;i++)
        fprintf(fp,"%d",lst->ventana->vec2[i]);
}
   fclose(fp);

}
/*-----------------------------------------------------------------*/
/* Depuracion: Imprime el contenido de una lista, mostrando su       */
/*            longitud y la posicion de la ventana                 */
/* Complejidad:       O(N)                                        */

void impLista(Lista lst)
{   pListaNodo p=lst->primero;

    /* Imprime la longitud */
    printf("\n[%d] <",lst->longitud);
```

```
    /* Recorre la lista imprimiendo sus elementos */
    while(p!=NULL)
    { /* Indica la posicion de la ventana */
          if (p==lst->ventana)
            printf("[ ");
          printf("%d ",p->info1);
          /* Indica la posicion de la ventana */
          if (p==lst->ventana)
            printf("] ");
          p=p->sig;
    }

    printf(">");
    /* Indica si la ventana se encuentra indefinida */
    if (lst->ventana==NULL)
       printf(" *");
    printf("\n");
}
```

## Anexo B. Programa de encriptación del esquema de construcción de un cifrador desde una permutación pseudo-aleatoria

```
/*
        Encriptar.c

        La rutina que prueba el PRNG (Yarrow-160) utiliza los comandos de
usuario.
*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <windows.h>
#include "yarrow.h"
#include "usersources.h"
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "Lista2.h"

/*  ESTE PROGRAMA UTILIZA LAS RUTINAS PUBLICAS DE YARROW-160  */
HANDLE ghInst;
HWND ghWndMain;
Lista lst;
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
/* Este programa tiene como entrada un archivo de texto */
{
        BYTE *buf;
```

```
    int vector[1000];
    BYTE str1[15],origen[15],destino[15],str2[15];
FILE  *ptrorigen, *ptrdestino, *ptcontrol,*ptranal;
    char unsigned reg,dato1,dato2,temp1,temp2;
int i,cont,num1,num2,indice1,indice2,res1,res2,operando1,operando2,k;
    srand( (unsigned)time( NULL ) );
    /* GENERO UN VECTOR DE NUMEROS ALEATORIOS */
    for (i=0; i < 1000; i++)
      vector[i]=rand()%128;
    lst=inicLista();
    buf = (BYTE*)malloc(4096);
    memset(buf,0x11,4096);
    prngInput(buf,4096,USERSOURCE1,1);
    memset(buf,0x22,4096);
    prngInput(buf,4096,USERSOURCE1,1);
    memset(buf,0x33,4096);
    prngInput(buf,4096,USERSOURCE1,1);
    free(buf);
    buf = (BYTE*)malloc(8192);
    memset(buf,0x44,8192);
    prngInput(buf,8192,USERSOURCE1,1);
    free(buf);
    buf = (BYTE*)malloc(20000);
    memset(buf,0x55,20000);
    prngInput(buf,20000,USERSOURCE1,1);
    prngAllowReseed(100);
    prngOutput(buf,13);
    prngOutput(buf,13);
    prngOutput(buf,13);
    prngOutput(buf,13);
```

```
   prngStretch(buf,13,str1,15);

   prngInput(buf,13,USERSOURCE1,5);
   prngAllowReseed(100);
   prngOutput(buf,13);
   free(buf);
   if ((ptranal=fopen("c:\\llaves.txt","wb")) == NULL){

                exit(1);
                                        }
   if ((ptcontrol=fopen("c:\\control.txt","w")) == NULL){

exit(1);
   }

   if ((ptrorigen=fopen("c:\\origen.txt","rb")) == NULL) {
exit(1);
   }
   if ((ptrdestino=fopen("c:\\destino.txt","w+b")) == NULL){

exit(1);
   }
   else {
      reg =255;
      k=0;
      lst=inicLista();
          do {
           srand( (unsigned)time( NULL ) );
           cont = fread(origen,sizeof(BYTE),15,ptrorigen);
           for (i=0; i < cont; i++)
```

```
            destino[i] = origen[i]^str1[i];
          num1=vector[k];
          num2=vector[k+1];
          indice1=num1/8;
          indice2=num2/8;
          res1=num1%8;
res2=num2%8;
          dato1=destino[indice1];
          dato2=destino[indice2];
operando1=1;
operando2=1;
          operando1=operando1<<res1;
          operando2=operando2<<res2;
          temp1=dato1&operando1;
          temp2=dato2&operando2;

 if (temp1!=temp2) {
                  destino[indice1]=dato1|operando1;
              destino[indice2]=dato2|operando2;
                       }

      prngInput(buf,20000,USERSOURCE1,1);
      prngAllowReseed(100);
      prngOutput(buf,13);
      prngOutput(buf,13);
      prngOutput(buf,13);
      prngOutput(buf,13);
      prngStretch(buf,13,str2,15);
           for (i=0; i < cont; i++)
            destino[i] = destino[i]^str2[i];
```

```
                for (i=0;i<cont ;i++)
                                        fprintf(ptranal,"%d",str1[i]);
                                    for (i=0;i<cont ;i++)
                                        fprintf(ptranal,"%d",str2[i]);

              anxLista(lst,num1,num2,str1,str2);
                    for (i=0; i < cont; i++)
                      fprintf(ptrdestino,"%d",destino[i]);
                    k=k+2;;
                 }
              while (! feof(ptrorigen));
         salvarLista(lst,ptcontrol);

        fclose(ptrorigen);

                fclose(ptrdestino);
        fclose(ptranal);

   }

        fclose(ptcontrol);

        destruirLista(lst);

return prngOutput(str2,15);
/* la salida es un archivo cifrado , un archivo con
   as secuencias de claves(analisis)  y un archivo para la
        FUNCION INVERSA*/
 }
```

## Anexo C. Programa que desencripta archivos cifrados bajo el esquema de cifrador basado en una permutación y PRNG Yarrow-160

```
/* Programa que se utiliza para desencriptar archivos cifrados */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <windows.h>
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "Lista2.h"
Lista lst;
int main(void)
{
 int
longit,infox,infoy,cont,i,k,reg,num1,num2,indice1,indice2,res1,res2,operando1,oper
ando2,temp1,temp2;
    byte datox[15],datoy[15],datos1[15],destino[15],dato1,dato2,vecx[15],vecy[15];
    FILE  *ptrorigen,*ptrdestino,*fp;

        if ((fp=fopen("c:\\control.txt","rb")) == NULL){

      exit(1);
        }
        else {
                    primLista(lst);
         fscanf(fp,"%d",&longit);
         for(;longit>0;longit--){
```

```
        fscanf(fp,"%d",&infox);
        fscanf(fp,"%d",&infoy);
        for (i=0;i!=15;i++)
              fscanf(fp,"%d",&vecx[i]);
        for (i=0;i!=15;i++)
              fscanf(fp,"%d",&vecy[i]);
                        anxLista(lst,infox,infoy,vecx,vecy);

                      }
    }


    fclose(fp);
    if ((ptrdestino=fopen("c:\\destino.txt","rb")) == NULL){

exit(1);
    }

    if ((ptrorigen=fopen("c:\\origen1.txt","w+b")) == NULL) {
exit(1);

    }
              else {
                        reg =255;
                 k=0;
                primLista(lst);
           do {
           cont = fread(datos1,sizeof(BYTE),15,ptrdestino);
           num1=lst->ventana->info1;
                                num2=lst->ventana->info2;
```

```
                    for (i=0; i < 15; i++)
                    datox[i]=lst->ventana->vec1[i];
                    for (i=0; i < 15; i++)
                    datoy[i]=lst->ventana->vec2[i];
                    for (i=0; i < cont; i++)
                    destino[i] = datos1[i]^datox[i];
                    indice1=num1/8;
          indice2=num2/8;
          printf("%d",indice1);
                    res1=num1%8;
res2=num2%8;
          dato1=destino[indice1];
          dato2=destino[indice2];
operando1=1;
operando2=1;
          operando1=operando1<<res1;
          operando2=operando2<<res2;
          temp1=dato1&operando1;
          temp2=dato2&operando2;

 if (temp1!=temp2) {
                    destino[indice1]=dato1|operando1;
              destino[indice2]=dato2|operando2;
                         }

 for (i=0; i < cont; i++)
            destino[i] = destino[i]^datoy[i];
                         for(i=0;i<cont;i++)
  fprintf(ptrorigen,"%c",destino[i]);
```

```
                sigLista(lst);

                                    }
                while (! feof(ptrdestino));

                  }
return 1;
}
```

## Anexo D. Programa que produce cinco pruebas estadísticas que son usadas para determinar si una secuencia binaria posee algunas características de ser secuencia verdaderamente aleatoria.

```
#include "stdio.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <windows.h>
#include "stdio.h"
#include "stdlib.h"
#include "math.h"


typedef unsigned char bytes;
FILE *ptranal, *test;
char ch;
int cad[8],seq[120];
int chi,e1,e2;
int i,dato,j,d,cont1,cont2,cont3,cont4,cont5,cont6,cont7,cont8,ques,n,num,k,n0,n1;
bytes vec1[15],vec2[15],uno,dos,pasa,copia[1];
int potencia(int base , int exp){
int p;
i=1;
p=1;
while (i<=exp){
p=base*p;
i++;
}
```

```
return p;

}
int unoocero(bytes copia, int k){
        uno=copia>>1;
   dos=uno<<1;

        if((copia^dos)==1)
                return 1;
        else
                return 0;


}
void testmonobit(){
cont1=0;
cont2=0;
chi=0;
n=0;
fprintf(test,"Secuencia %d\n ", num);
i=0;
k=0;
while (i<15) {

       j=8;
                while(j>0){
                        copia[0]=vec1[i];
          ques=unoocero(*copia,i);
                        vec1[i]=vec1[i]>>1;
                        if (ques == 1) {
```

```
            cont1++;
                        cad[j-1]=1;

                       }
           else {
                              cont2++;
                              cad[j-1]=0;

                       }
                         n++;
                              j--;

       }
                 for(j=0;j!=8;j++){
                   fprintf(test,"%d",cad[j]);
         seq[k]=cad[j];
                   k=k+1;
                 }
                  i++;
                 }

     }
void testpokerbit(){
k=2;
cont1=0;
cont2=0;
cont3=0;
cont4=0;
cont5=0;
cont6=0;
```

```
cont7=0;
cont8=0;
n=0;
while (k<=120){

        if((seq[k-2]==0)&&(seq[k-1]==0)&&(seq[k]==0))
                cont1++;
        if((seq[k-2]==0)&&(seq[k-1]==0)&&(seq[k]==1))
                cont2++;
        if((seq[k-2]==0)&&(seq[k-1]==1)&&(seq[k]==0))
                cont3++;
        if((seq[k-2]==0)&&(seq[k-1]==1)&&(seq[k]==1))
                cont4++;
        if((seq[k-2]==1)&&(seq[k-1]==0)&&(seq[k]==0))
                cont5++;
        if((seq[k-2]==1)&&(seq[k-1]==0)&&(seq[k]==1))
                cont6++;
        if((seq[k-2]==1)&&(seq[k-1]==1)&&(seq[k]==0))
                cont7++;
        if((seq[k-2]==1)&&(seq[k-1]==1)&&(seq[k]==1))
                cont8++;
        n++;
k++;
}
}
void testparbit(){
k=1;
cont1=0;
cont2=0;
cont3=0;
```

```
cont4=0;
n=0;
while (k<=120){

        if((seq[k-1]==0)&&(seq[k]==0))
                cont1++;
        if((seq[k-1]==0)&&(seq[k]==1))
                cont2++;
        if((seq[k-1]==1)&&(seq[k]==0))
                cont3++;
   if((seq[k-1]==1)&&(seq[k]==1))
                cont4++;
        n++;
k++;
}

}
void testautocorrebit(){
k=1;
cont1=0;
n=0;
d=3;
while (k<=120-d-1){

        cont1=cont1+seq[k]^seq[k+d];
        n++;
k++;
}

}
```

```
void testrunbit(){
k=1;
i=3;
k=2;

cont1=0;
cont2=0;
cont3=0;
cont4=0;
n=0;
while (k<=120){

        if((seq[k-2]==1)&&(seq[k-1]==0)&&(seq[k]==1))
                cont1++;
        if((seq[k-2]==0)&&(seq[k-1]==1)&&(seq[k]==0))
                cont3++;
                n++;
k++;
}
k=3;
while (k<=120){

        if((seq[k-3]==1)&&(seq[k-2]==0)&&(seq[k-1]==0)&&(seq[k]==1))
                cont2++;
        if((seq[k-3]==0)&&(seq[k-2]==1)&&(seq[k-1]==1)&&(seq[k]==0))
                cont4++;

                n++;
k++;
}
```

```
}
int main(void){

if ((test=fopen("c:\\prueba.txt","wb")) == NULL){

                    exit(1);
                                        }
if ((ptranal=fopen("c:\\llaves.txt","rb")) == NULL){

                    exit(1);
                                        }
 else
 {

num=1;
 do{

 for(i=0;i!=15;i++){
     fscanf(ptranal,"%d\n",&dato);
          vec1[i]=dato;
               vec2[i]=dato;
 }


 /* Aqui se inicia el llamado del test de frecuencia monobit */

  testmonobit();
  fprintf(test,"\n");
  fprintf(test,"Test de Frecuencia\n");
  fprintf(test,"Numero de 1 's =%d\n",cont1);
```

```
  n1=cont1;
  fprintf(test,"Numero de 0 's =%d\n",cont2);
  n0=cont2;
  chi=(cont2-cont1)*(cont2-cont1)/n;
  fprintf(test,"estadistico chi cuadrado %3.2f\n",chi);

  testparbit();
  fprintf(test, " Test Serial \n");
  fprintf(test,"Numero de 0´s y 0's = %d\n",cont1);
  fprintf(test,"Numero de 0's y 1's=%d\n",cont2);
  fprintf(test,"Numero de 1 's y 0's =%d\n",cont3);
  fprintf(test,"Numero de 1 's y 1's = %d\n",cont4);
  chi=4*(cont1*cont1+cont2*cont2+cont3*cont3+cont4*cont4)/(n-1)-
(n0*n0+n1*n1)*2/n;
  fprintf(test,"estadistico chi cuadrado %3.2f\n",chi);
  testpokerbit();
  fprintf(test, " Test Poker \n");
  fprintf(test,"Numero de 0´s y 0's y 0's = %d\n",cont1);
  fprintf(test,"Numero de 0´s y 0's y 1's = %d\n",cont2);
  fprintf(test,"Numero de 0´s y 1's y 0's = %d\n",cont3);
  fprintf(test,"Numero de 0´s y 1's y 1's = %d\n",cont4);
  fprintf(test,"Numero de 1´s y 0's y 0's = %d\n",cont5);
  fprintf(test,"Numero de 1´s y 0's y 1's = %d\n",cont6);
  fprintf(test,"Numero de 1´s y 1's y 0's = %d\n",cont7);
  fprintf(test,"Numero de 1´s y 1's y 1's = %d\n",cont8);

chi=8*(cont1*cont1+cont2*cont2+cont3*cont3+cont4*cont4+cont5*cont5+cont6*co
nt6+cont7*cont7+cont8*cont8)/40-40;
  fprintf(test,"estadistico chi cuadrado %3.2f\n",chi);
  fprintf(test, " Run Test  \n");
```

```
  fprintf(test,"Numero de gaps de longitud 1 = %d\n",cont1);
  fprintf(test,"Numero de bloques de longitud 1 = %d\n",cont3);
  fprintf(test,"Numero de gaps de longitud 2 = %d\n",cont2);
  fprintf(test,"Numero de bloques de longitud 2 = %d\n",cont4);
  i=1;
  e1=(120-i+3)/potencia(2,i+2);
  i=2;
  e2=(120-i+3)/potencia(2,i+2);
  chi=(cont3-e1)*(cont3-e1)/e1+(cont4-e2)*(cont4-e2)/e2+(cont1-e1)*(cont1-
e1)/e1+(cont3-e2)*(cont3-e2)/e2;
  fprintf(test,"estadistico chi cuadrado %3.2f\n",chi);
  testautocorrebit();
  fprintf(test, "Test de correlacion  \n");
fprintf(test,"Numero de bits no iguales a sus d=3 mas delante = %d\n",cont1);
chi=2*(cont1-(120-d)/2)*(sqrt(n-d));
fprintf(test,"estadistico chi cuadrado %3.2f\n",chi);
 num=num+1;
 }

while (! feof(ptranal));
 }
 printf("termina prueba estadistica");
          ch=getchar();
 return 1;
}
```