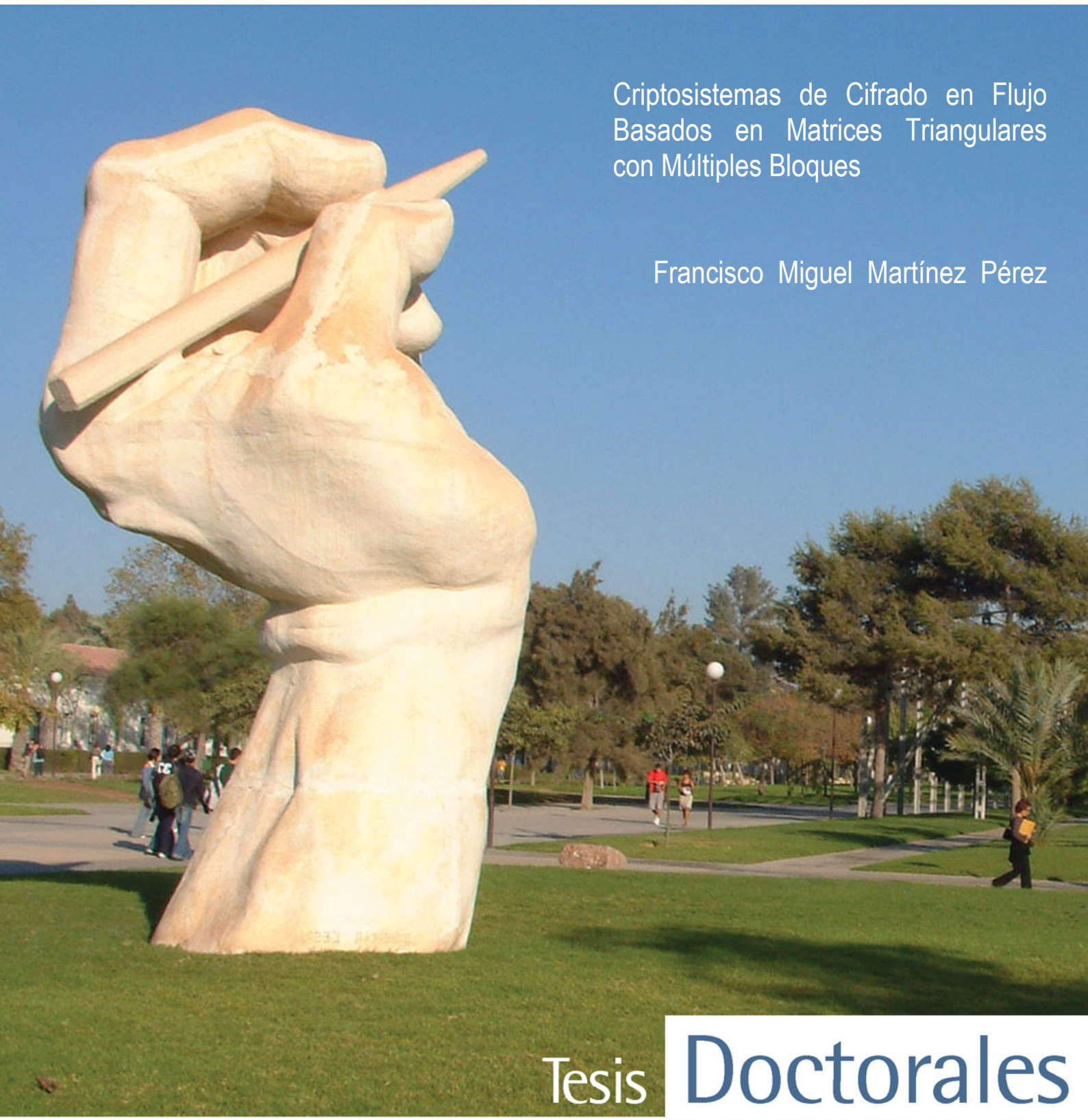
Universitat d'Alacant
Universidad de Alicante

Criptosistemas de Cifrado en Flujo Basados en Matrices Triangulares con Múltiples Bloques

Francisco Miguel Martínez Pérez



Tesis Doctorales

www.eltallerdigital.com

UNIVERSIDAD de ALICANTE

Departamento de Ciencia de la Computación
e Inteligencia Artificial

Escuela Politécnica Superior de Alicante

Criptosistemas de Cifrado en Flujo Basados en Matrices Triangulares con Múltiples Bloques

Universidad de Alicante

Francisco Miguel Martínez Pérez

Tesis presentada para aspirar al grado de
Doctor por la Universidad de Alicante
en Ingeniería Informática y Computación

Dirigida por:
Dr. Rafael Ignacio Álvarez Sánchez
Dr. Antonio Zamora Gómez

Universitat d'Alacant
Universidad de Alicante

Resumen

En este trabajo se diseña y analiza un generador pseudoaleatorio basado en matrices triangulares superiores de 3×3 bloques con elementos en $\mathbb{Z}_p$, siendo p primo; adecuado para ser empleado como generador de secuencia cifrante en un criptosistema de cifrado en flujo binario aditivo. Con ese objetivo se han buscado los parámetros adecuados (primo, polinomios primitivos y tamaños de bloque) para garantizar un adecuado nivel de seguridad criptográfica.

Aunque finalmente se ha propuesto un generador especifico con una parametrización concreta, se debe enfatizar que en realidad se trata de una familia de generadores que se pueden adecuar, sin pérdida de aleatoriedad ni aumento de coste computacional apreciable, a distintos requisitos impuestos por la aplicación, implementación, arquitectura, etc., con sólo variar el parámetro d de la versión optimizada.

Dentro de esta familia cabe resaltar que, haciendo uso de un tipo específico de trinomio primitivo, se han logrado rendimientos competitivos con los estándares actuales de cifrado en flujo como RC4, Salsa20, AES-OFB o HC128. Dado el rendimiento alcanzado por el generador optimizado finalmente propuesto, resulta perfectamente adecuado para su uso en protocolos, sistemas y aplicaciones que requieran de criptografía simétrica. Además, considerando algunas características de seguridad adicionales, como el soporte directo de tamaños de clave variable, puede ser interesante como sustituto de otros algoritmos que han sido considerados inseguros recientemente, como RC4.

El generador propuesto tiene la característica de proporcionar una extracción de 64 bytes por iteración, que resulta apropiada para su implementación tanto en software como en hardware o, incluso, en hardware reconfigurable (FPGA).

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Abstract

In this work we design and analyze a pseudorandom generator based on 3×3 block upper triangular matrices with elements in $\mathbb{Z}_p$, where p is prime; it is suitable for its use as a key-stream generator in a binary additive stream cipher. With this aim, we have sifted to find the appropriate parameters (prime, primitive polynomials and block sizes) in order to ensure adequate level of cryptographic security.

It must be remarked that, although we have proposed a specific generator with a concrete parameterization, it is really a family of generators that can be adapted, without loss of randomness or significant computational cost increase, to different requirements regarding application, implementation, architecture, etc., just by simply varying the d parameter in the optimized version.

Considering this family, it is worth noting that, using a specific type of primitive trinomial, the generator has attained comparable results to current standard stream ciphers like RC4, Salsa20, AES-OFB or HC128. Given the performance achieved by the proposed optimized generator, it is well suited for its use in protocols, systems and applications that require symmetric cryptography. Moreover, considering some other additional security features, such as the direct support for variable key lengths, it can be an interesting substitute for other algorithms that have been deemed insecure recently, like RC4.

The proposed generator has an extraction mechanism that provides 64 bytes of data per iteration, which is ideal for its implementation in software, hardware or even in reconfigurable hardware (FPGA).

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Agradecimientos

Me gustaría dar mi más sincero agradecimiento a los doctores Rafael Álvarez y Antonio Zamora, por su apoyo y dedicación, desde el principio con este trabajo conjunto; gracias por vuestra paciencia, por vuestras enseñanzas, gracias en definitiva por confiar en mí.

A todos los miembros del Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, en ellos siempre he encontrado ánimo y motivación, además de amistad.

A Mamen, Fran y Blanca, son una parte de mí, este trabajo es de ellos también.

A mis padres, a mis hermanas, a mis amigos; a todos ellos, en agradecimiento, se lo dedico.

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Índice de contenidos

Resumen III
Abstract V
Agradecimientos VII
Índice de contenidos IX
Introducción 1
Motivación 1
Objetivo 1
Contribuciones 2
Estructura 3
1 Preliminares 5
1.1 Conceptos básicos 5
1.2 Generadores aleatorios 9
1.2.1 Generadores realmente aleatorios 10
1.2.2 Generadores pseudoaleatorios 11
1.2.2.1 Generadores congruenciales 12
1.2.2.2 Registros de desplazamiento con retroalimentación 13
1.3 Medida de la aleatoriedad 15
1.3.1 Postulados de Golomb 15
1.3.2 Métricas estadísticas 17
1.3.3 Complejidad lineal y su perfil (Berlekamp-Massey) 19
1.3.4 Análisis de aleatoriedad 20
1.3.4.1 PractRand 20
1.3.4.2 TestU01 21
1.4 Cifrado en flujo 21
1.4.1 Características del cifrado en flujo y en bloque 21

Universitat d'Alacant
Universidad de Alicante

1.4.2 Cifrado de Vernam 23
1.4.3 Cifradores en flujo síncronos y autosincronizantes 25
1.5 Generadores con los que se compara 26
1.5.1 AES en modo OFB 26
1.5.2 Blum-Blum-Shub 27
1.5.3 RC4 28
1.5.4 Salsa20 29
1.5.5 HC128 30
2 Matrices triangulares superiores por bloques 33
2.1 Propiedades algebraicas 33
2.2 Potencias de matrices TSB 36
2.3 Orden de las matrices TSB 40
3 Cifrado en flujo basado en matrices triangulares superiores por bloques . 43
3.1 Generador pseudoaleatorio basado en matrices TSB de 3 × 3 bloques 43
3.1.1 Restricciones a los parámetros 45
3.2 Primera aproximación 49
3.2.1 Análisis de los test 54
3.2.2 Rendimiento 55
3.3 Mejora del sistema de extracción 56
3.3.1 Diseño de la segunda batería de experimentos 58
3.3.2 Variante en los sistemas de extracción. 63
3.4 Mejoras en el filtrado 64
3.4.1 Cajas de sustitución 65
3.4.2 Ventajas del filtro *s-box* 32 × 32 67
3.4.3 Batería de experimentación 67
3.5 Optimización el algoritmo 68
3.5.1 Enfoque optimizado 70
3.5.2 Resultados 74
4 Conclusiones 77
4.1 El generador propuesto 77
4.1.1 Posibilidades de uso 78
4.2 Contribuciones 78

Universitat d'Alacant
Universidad de Alicante

4.2.1 Extensión a 3 × 3 bloques 78
4.2.2 Estudio y parametrización 79
4.2.3 Bloques de tamaño 1 × 1 79
4.2.4 Extracción con aprovechamiento máximo 79
4.2.5 Tablas de polinomios primitivos 79
4.2.6 Inclusión de filtrado con *s-box* 32 × 32 79
4.2.7 Algoritmo optimizado 80
4.2.8 Publicaciones 80
4.3 Propuestas futuras 80
4.3.1 Otros primos 80
4.3.2 Implementaciones específicas 81
4.3.3 Nuevos planteamientos con 2 × 2 bloques 81
4.3.4 Búsquedas de polinomios primitivos 82
4.3.5 Extensión a más bloques 82
4.3.6 Criptoanálisis 82
Bibliografía 83
Índices y acrónimos 91
Apéndice A Baterías de experimentación 95
A.1 Primera batería 95
A.2 Segunda batería 120
A.3 Tercera batería 133
A.4 Cuarta batería 136
Apéndice B Códigos fuente 159
B.1 Generador para distintos primos 160
B.2 Generador con caja de sustitución 183
B.3 Generador óptimo 199
Apéndice C Polinomios primitivos 211

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Introducción

Motivación

Es evidente que la motivación para un trabajo como este se encuentra en la demanda constante de mantener niveles aceptables de seguridad en la comunicación. Esa demanda se debe a una sociedad actual basada en la comunicación y el manejo de la información. Las formas de interactuar socialmente (transacciones comerciales, relaciones personales, trabajo, relaciones con la administración pública, etc.) han cambiado, y demandan canales de comunicación cada vez más seguros.

El enfrentamiento constante que existe en esta sociedad de la comunicación entre la seguridad informática y la ciberdelincuencia es llevada al plano de la investigación por la criptología, que enfrenta a criptógrafos y criptoanalistas en un intento de adelantarse a los acontecimientos y necesidades demandadas. Esa sensación de desafío y competición, unida a una inmensa utilidad práctica, es lo que hace de la criptología una actividad académica e investigadora fascinante.

Universidad de Alicante

Nuestro trabajo se ha realizado con la ilusión de que el criptosistema resultante de nuestro trabajo pudiera sustituir un estándar en cifrado en flujo como RC4, que recientemente ha sido prohibido para su uso con el protocolo TLS. Con la misma ilusión, esperamos que futuros trabajos criptoanalicen nuestro criptosistema propuesto, y así continuar este juego que la criptología propone.

Objetivo

El objetivo de este trabajo ha sido obtener un cifrador en flujo a partir de las excelentes propiedades criptográficas y de aleatoriedad observadas en ciertas construcciones de matrices triangulares superiores por bloques con elementos en $\mathbb{Z}_p$, siendo p primo. Para ello se ha diseñado un generador pseudoaleatorio, como base de dicho cifrador, a partir de las propiedades de las matrices referidas.

La pretensión añadida de que el criptosistema propuesto resultase competitivo frente a estándares actuales de prestigio como AES, RC4, Salsa20 o HC128, ha sido también finalmente satisfecha tras el planteamiento de un diseño optimizado.

Contribuciones

Este trabajo es continuación de otros anteriores sobre matrices triangulares superiores por bloques, exclusivamente de 2×2 bloques. La primera contribución que se ha realizado es la extensión a matrices de 3×3 bloques, aportando el correspondiente fundamento matemático en el que se ha basado la construcción de los generadores pseudoaleatorios estudiados.

El estudio de estos generadores se ha realizado sobre su parametrización. La búsqueda de los parámetros más adecuados ha proporcionado interesante información sobre el comportamiento de estos parámetros tras el análisis de los resultados de las exhaustivas pruebas realizadas.

El uso de bloques de tamaño 1×1, sistemas de extracción más productivos y un filtrado no lineal basado en una *s-box* de 32×32 son otras de las novedades que se han aplicado en este trabajo y que lo hacen distinguirse de los anteriores. Las tres nuevas aportaciones han resultado efectivas en sus objetivos, mejorando los resultados en los generadores pseudoaleatorios probados.

Un enfoque distinto al aplicado hasta ahora con las matrices triangulares superiores por bloques, en la implementación, ha reducido significativamente el coste del algoritmo, usando un tipo específico de trinomios primitivos. Esta novedosa aportación, sumada a la ya indicada de usar un sistema de extracción más productivo, ha permitido alcanzar el objetivo anteriormente indicado, de obtener un generador pseudoaleatorio competitivo frente a los actuales estándares.

Además de los artículos que se han publicado a lo largo del periodo de investigación [14-17], durante este mismo periodo, para poder construir los generadores pseudoaleatorios y realizar las pruebas, se ha realizado también una labor de búsqueda de polinomios primitivos (necesarios para su implementación), que ha culminado con la elaboración de una tabla, que aunque no fuera objetivo directo de este trabajo, se ha considerado interesante incluir también como contribución, debido al interés que suscita este tipo de polinomios, por sus propiedades, en criptografía y otras disciplinas.

Estructura

El contenido de este trabajo se ha distribuido en cuatro capítulos.

El capítulo 1, preliminares, pretende servir de referencia a aquellos que no están del todo familiarizados con algunas cuestiones tratadas. A su vez, se estructura en una primera sección de conceptos básicos, otra sobre generadores aleatorios y otra, posterior, sobre las medidas de la aleatoriedad para éstos; después otra sección sobre el cifrado en flujo y dos más con ejemplos de cifradores que posteriormente se usarán de referencia en alguna comparativa.

El capítulo 2 contiene toda la base matemática que sustenta al generador pseudoaleatorio. Se parte de un teorema genérico sobre las propiedades del producto de matrices triangulares superiores por bloques. En una segunda sección se concretan las fórmulas para las potencias de estas matrices en los casos de 2×2 bloques y 3×3 bloques con dos teoremas más. Finalmente, en una tercera sección, se propone la construcción de generadores pseudoaleatorios, cuyo periodo coincide con el orden de las matrices, a partir de polinomios primitivos.

En el capítulo 3 se ha detallado la evolución del proceso de estudio y desarrollo del criptosistema. En una primera sección se ha descrito cómo es el diseño de partida y sus restricciones. Posteriormente se indica cómo se ha diseñado la primera batería de pruebas. Las conclusiones extraídas llevan al trabajo comentado en la tercera sección, donde se mejora el sistema de extracción en una segunda batería de pruebas. La tercera batería es consecuencia de las mejoras en el sistema de filtrado que se exponen en una cuarta sección. La optimización del algoritmo, test y comparativa aparecen en una última sección.

Universitat d'Alacant
Universidad de Alicante

Las conclusiones en el capítulo 4 incorporan, además de una propuesta concreta de generador pseudoaleatorio, un detalle de las contribuciones y una serie de propuestas futuras.

Finalmente se ha incluido en apéndices los resultados de las pruebas realizadas, el código fuente que se ha considerado relevante y la tabla de polinomios primitivos.

Universitat d'Alacant
Universidad de Alicante

1 Preliminares

1.1 Conceptos básicos

La criptografía hace años que dejó de ser un arte para convertirse en un conglomerado de técnicas sobre protección de la información frente a observadores no autorizados. Uno de trabajos fundamentales sobre los que se apoya prácticamente toda la actual teoría criptográfica fue desarrollado por Claude Elwood Shannon que describió a finales de los años cuarenta en su teoría matemática de la comunicación [91]. Poco después, en colaboración con Warren Weaver [92], estableció su esquema de comunicación (figura 1.1).

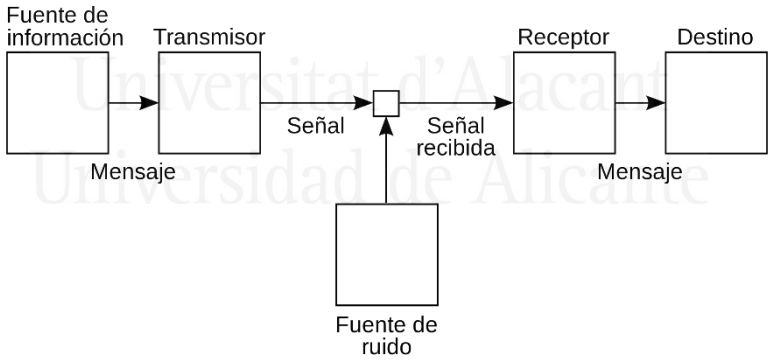


Figura 1.1: Esquema original de comunicación de Shannon

En este esquema, el transmisor y receptor son entidades (persona física, terminal, servidor, agente software, hardware, etc.) que envían, reciben o manipulan información. Desde el punto de vista criptográfico interesa matizar las entidades que participan en la comunicación en función de la relación de legitimidad que poseen con el mensaje.

Definición 1.1: Emisor

El emisor es aquella entidad que transmite de forma legítima un mensaje.

Universitat d'Alacant
Universidad de Alicante

Definición 1.2: Receptor

El receptor es una entidad que recibe un mensaje estando autorizado para ello.

Definición 1.3: Atacante

Un atacante es la entidad que intenta acceder al mensaje sin ser el emisor o el receptor.

Otro concepto que interesa matizar desde el punto de vista criptográfico es el canal.

Definición 1.4: Canal

Un canal es un medio capaz de transmitir un mensaje de una entidad a otra.

La mayoría de los canales de comunicación, o al menos aquellos que hoy en día despiertan nuestro interés (Internet, WiFi, telefonía, etc.), no son seguros.

Definición 1.5: Canal seguro

Un canal seguro es un canal sobre el que un atacante no puede realizar operaciones de lectura, escritura, borrado o reordenación.

La finalidad última de la criptografía, cuya etimología está en la unión de los términos griegos *κρυπτός* (oculta) y *γραφία* (escritura), es la creación de canales seguros, es decir, convertir información perfectamente comprensible en un formato completamente ilegible para un observador que carezca de cierta información secreta [64].

Definición 1.6: Criptografía

La criptografía es el estudio de las técnicas matemáticas relacionadas con aspectos de la seguridad de la información tales como la confidencialidad, integridad, autentificación y no repudio.

La confidencialidad, secretismo o privacidad, consiste en garantizar que el contenido del mensaje de una comunicación quede limitado únicamente a las entidades autorizadas.

La integridad de los datos consiste en evitar la manipulación o alteración no autorizada de los datos; por autentificación se entiende la correcta identificación de todas las entidades que participan en la comunicación; mientras que el no repudio consiste en evitar que alguna de las entidades de la comunicación niegue haber realizado ciertas acciones.

Los elementos de un sistema criptográfico se definen sobre un conjunto finito de símbolos o alfabeto. El más común en la criptografía moderna es el binario: $\{0, 1\}$, ya que cualquier otro alfabeto se puede, en última instancia, codificar en binario.

Definición 1.7: Espacio de mensajes

El conjunto $\mathcal{M}$ de todas las cadenas de símbolos posibles formadas combinando los elementos de un alfabeto de definición, se denomina el espacio de mensajes. Se denomina texto en claro a cada elemento de $\mathcal{M}$.

Definición 1.8: Espacio de criptogramas

Un espacio de criptogramas $\mathcal{C}$ consiste en un conjunto de cadenas de símbolos formadas combinando los elementos de un alfabeto de definición. Cada elemento de $\mathcal{C}$ se denomina texto cifrado o criptograma.

Definición 1.9: Espacio de claves

El espacio de claves $\mathcal{K}$ es en un conjunto de cadenas de símbolos formadas combinando los elementos de un alfabeto de definición, donde cada elemento de $\mathcal{K}$ se denomina clave.

Definición 1.10: Función de cifrado

Para cada $e \in \mathcal{K}$, E_e denota la función que asocia de forma única un elemento de $\mathcal{C}$ con un elemento de $\mathcal{M}$ y se denomina función de cifrado o transformación de cifrado. El proceso que consiste en aplicar la transformación E_e a un texto en claro $m \in \mathcal{M}$ se denomina cifrado de m.

Universitat d'Alacant
Universidad de Alicante

Definición 1.11: Función de descifrado

Para cada $d \in \mathcal{K}$, D_d denota la función que asocia de forma única un elemento de $\mathcal{M}$ con un elemento de $\mathcal{C}$ y se denomina función de descifrado o transformación de descifrado. El proceso que consiste en aplicar la transformación D_d a un texto cifrado $c \in \mathcal{C}$ se denomina descifrado de c.

Es absolutamente necesario que las asociaciones entre los conjuntos $\mathcal{M}$ y $\mathcal{C}$ definidas por las funciones E_e y D_d sean biyecciones para que el proceso se pueda invertir. De esta forma, con una misma clave, se obtiene un único texto cifrado para cada texto en claro y un único texto en claro para cada texto cifrado.

Definición 1.12: Criptosistema

Un criptosistema consiste en un conjunto $\{E_e : e \in \mathcal{K}\}$ de funciones de cifrado y el correspondiente conjunto $\{D_d : d \in \mathcal{K}\}$ de funciones de descifrado, con

la propiedad de que para cada $e \in \mathcal{K}$ existe una única clave $d \in \mathcal{K}$ tal que $D_d(E_e(m)) = m$ para todo $m \in \mathcal{M}$.

Otros términos para referirse a un criptosistema son cifrador, primitiva, algoritmo de cifrado, algoritmo criptográfico o, simplemente, cifra. El objetivo de este trabajo ha sido el diseño y estudio de un tipo de criptosistemas basados en matrices triangulares superiores por bloques.

Definición 1.13: Criptoanálisis

El criptoanálisis es el estudio de las técnicas matemáticas para contrarrestar, o anular, la criptografía.

El objetivo del criptoanálisis es obtener de un texto cifrado el texto en claro (o una parte) sin poseer el secreto necesario, es decir la clave. Cuando este objetivo se puede cumplir en un tiempo computacionalmente admisible, se dice que el algoritmo o criptosistema se ha roto.

Existe discrepancia en la literatura a la hora de considerar como criptoanálisis la técnica más elemental, denominada ataque de fuerza bruta, consistente en probar todas las combinaciones posibles de clave; ya que, aunque cumple con el objetivo de obtener el texto en claro, no se apoya en ninguna otra técnica que contribuya a reducir la búsqueda de la clave o la obtención directa del texto en claro. A pesar de ello, cualquier criptosistema debe protegerse de esta técnica usando un espacio de claves de tamaño adecuado (suficientemente grande) a la capacidad computacional del momento.

Universitat d'Alacant
Universidad de Alicante

Definición 1.14: Criptología

La criptología es la ciencia que comprende, conjuntamente, el estudio de la criptografía y del criptoanálisis.

Algunos autores consideran también criptología la esteganografía: práctica consistente en ocultar mensajes u otros objetos en otros portadores con idea de que pasen desapercibidos; y a su antagónica, el esteganoanálisis.

Definición 1.15: Criptosistema simétrico

Se dice que un criptosistema formado por los conjuntos de transformaciones de cifrado $\{E_e : e \in \mathcal{K}\}$, y de descifrado $\{D_d : d \in \mathcal{K}\}$, es un criptosistema simétrico si se puede determinar d a partir de e con un coste computacional bajo.

En la mayoría de criptosistemas simétricos ambas claves coinciden, en otros, como por ejemplo en el caso de una matriz y su inversa, se tratan de un valor y su opuesto o simétrico. En cualquier caso, los criptosistemas simétricos

son aquellos en los que la clave de descifrado se puede obtener a partir de la de cifrado y viceversa de una forma relativamente fácil, por lo que sólo es necesario conocer una de ellas.

Definición 1.16: Criptosistema asimétrico

Se dice que un criptosistema formado por los conjuntos de transformaciones de cifrado $\{E_e: e \in \mathcal{K}\}$, y de descifrado $\{D_d: d \in \mathcal{K}\}$, es un criptosistema asimétrico si el coste computacional asociado a determinar d a partir de e es muy elevado.

En un criptosistema asimétrico, para cada una de las parejas de claves (e, d), la clave e se hace pública y la d se mantiene en secreto; ya que conocer e, que se denomina clave pública, no permite en un tiempo computacionalmente admisible, conocer su pareja d, que se denomina clave privada.

Los protocolos criptográficos que proporcionan servicios de integridad, autenticación y no repudio combinan funciones hash, que son funciones que asocian cadenas binarias de tamaño fijo a cadenas binarias de tamaño arbitrario, con criptosistemas asimétricos. El criptosistema objeto de este trabajo es un criptosistema simétrico, orientado a la confidencialidad.

1.2 Generadores aleatorios

Universitat d'Alacant
Universidad de Alicante

La seguridad de muchos sistemas criptográficos está basada en el uso de números aleatorios; como pueden ser la secuencia cifrante de un cifrador en flujo o la clave secreta de un cifrador en bloque. Estos valores han de ser lo suficientemente impredecibles para que un atacante no sea capaz de averiguarlos mediante el uso de técnicas probabilísticas. La calidad de los valores aleatorios generados es de suma importancia, puesto que una clave fácil de predecir reduce la seguridad de los sistemas criptográficos.

Un generador aleatorio es un dispositivo o algoritmo que produce una secuencia de bits aleatoria con la que obtener estos valores impredecibles, entendiendo por secuencia de bits aleatoria a aquella en la que los bits no muestran sesgo ni autocorrelación significativa.

Definición 1.17: Sesgo

Se dice que una secuencia de bits contiene sesgo cuando la probabilidad de que un bit de dicha secuencia tenga un valor 1 sea significativamente mayor que la de que tenga un valor 0 o viceversa.

Definición 1.18: Autocorrelación

Cuando la probabilidad de que un bit determinado de una secuencia sea 1 o 0 dependa, de alguna forma, de los bits anteriores de la misma secuencia; se dice que los bits son estadísticamente dependientes y que la secuencia presenta una autocorrelación significativa.

1.2.1 Generadores realmente aleatorios

Algunos generadores se basan en fuentes naturales de aleatoriedad para obtener las secuencias de bits aleatorias.

Definición 1.19: Generador realmente aleatorio

Un generador realmente aleatorio es un dispositivo físico o un algoritmo que explota una fuente natural de aleatoriedad para proporcionar una secuencia de bits aleatoria.

Las fuentes naturales de aleatoriedad pueden ser de origen interno o externo a la computadora que implementa al generador. En este último caso se combinan uno o más fenómenos físicos como, por ejemplo, el tiempo de duración de las partículas en procesos radiactivos, el ruido térmico de una resistencia, el ruido de un micrófono o de una cámara, la inestabilidad de frecuencia de un oscilador, etc. Cuando el origen de la fuente se busca en la propia computadora se combinan recursos como el reloj del sistema, la frecuencia de pulsación de teclas, el contenido de buffers de entrada y salida, estadísticas del sistema operativo como la carga de procesador o el uso de memoria y, hasta, la latencia del disco duro o de los dispositivos de red [42].

Universidad de Alicante

Los generadores realmente aleatorios presentan ciertos inconvenientes para su uso criptográfico [66, 68, 89, 94]. Un atacante podría observar o manipular la fuente de aleatoriedad de forma que le permitiera predecir la secuencia con cierta probabilidad. Además se ha de monitorizar continuamente el funcionamiento de la fuente para evitar que deje de ser suficientemente aleatoria.

Otro problema es que la mayoría de las fuentes de aleatoriedad son defectuosas, bien sea porque el sesgo de la secuencia es significativo, o porque el valor de autocorrelación es excesivo. Existen diversas técnicas para eliminar o minimizar estas deficiencias y algunas emplean funciones hash criptográficas o cifradores en bloque.

Para determinadas aplicaciones criptográficas, como el cifrado tipo Vernam, resulta mucho más conveniente utilizar generadores pseudoaleatorios,

basados en algoritmos deterministas para generar la secuencia a partir de un valor inicial, que no presentan estos inconvenientes.

Las secuencias generadas de forma determinista de manera que, en la práctica, son indistinguibles de una secuencia realmente aleatoria se denominan secuencias pseudoaleatorias.

1.2.2 Generadores pseudoaleatorios

Definición 1.20: Generador pseudoaleatorio

Un generador pseudoaleatorio es un algoritmo determinista que, tomando una pequeña cantidad de bits aleatorios, genera una secuencia de bits pseudoaleatoria de mucha mayor longitud.

El uso de un algoritmo determinista implica que se genere siempre la misma secuencia a partir del mismo valor inicial o semilla.

Definición 1.21: Semilla

La entrada a un generador pseudoaleatorio consiste en una pequeña cantidad de bits aleatorios denominada semilla.

Los generadores pseudoaleatorios presentan la notable propiedad de ser amplificadores o extensores de aleatoriedad eficientes. A partir de un pequeño valor aleatorio: la semilla, que hace el papel de clave, producen secuencias muy largas que parecen ser aleatorias para cualquier observador. Por lo tanto, la salida de dicho generador pseudoaleatorio se puede utilizar en lugar de una secuencia realmente aleatoria en cualquier aplicación que requiera dicho tipo de secuencias. De esta forma, para compartir la secuencia generada basta con compartir la semilla.

Los beneficios de los generadores pseudoaleatorios en criptografía son innumerables dado que la implementación de la mayoría de los servicios criptográficos requiere abundante aleatoriedad de gran calidad.

Definición 1.22: Espacio de semillas

Un espacio de semillas de un generador pseudoaleatorio es el conjunto de posibles valores iniciales que el generador usa para generar las secuencias.

Para garantizar la seguridad de un generador pseudoaleatorio se debe elegir un tamaño de semilla lo suficiente grande como para que un atacante no pueda realizar una prueba exhaustiva de todas las semillas, lo que se denomina ataque por fuerza bruta, en un tiempo y con un coste razonables [51, 80].

Universidad de Alicante

Definición 1.23: Periodo de la secuencia

Toda secuencia pseudoaleatoria acaba por repetirse tras una determinada longitud que se denomina periodo de la secuencia.

Otro requisito de seguridad en un generador pseudoaleatorio es que el período de la secuencia generada debe ser lo suficientemente grande para que no se repita en la práctica. La longitud necesaria dependerá de la aplicación y de la frecuencia con que se cambie la semilla.

La principal propiedad que justifica el empleo de un generador pseudoaleatorio en criptografía es la impredecibilidad de la secuencia generada, lo que obliga a un atacante a realizar una búsqueda exhaustiva de semillas para obtener el resto de la secuencia.

Definición 1.24: Secuencia impredecible

Se dice que una secuencia es impredecible si no existe ningún algoritmo eficiente que, a partir de un trozo de dicha secuencia, sea capaz de determinar el siguiente bit con probabilidad significativamente superior a 1/2.

Se puede encontrar información más detallada acerca de los generadores aleatorios y pseudoaleatorios en [68, 89, 94].

1.2.2.1 Generadores congruenciales

Uno de los principales generadores pseudoaleatorios utilizados hoy día es el generador congruencial lineal o LCG (*Linear Congruential Generator*) [54]. Introducido por Lehmer [60] en 1951, es uno de los más antiguos y conocidos.

Universidad de Alicante

Definición 1.25: Generador congruencial lineal

Un generador congruencial lineal (LCG) es aquel que calcula una secuencia de números pseudoaleatoria con una ecuación lineal definida por la relación de recurrencia

$$X_n = (aX_{n-1} + b)\ mod\ m\ , \qquad \textbf{(1.1)}$$

donde a, b y m son constantes enteras y X_0 es el valor inicial o semilla de la secuencia X_n.

El período en esta clase de generadores nunca es superior a m; en el mejor de los casos, cuando a, b y m se escogen de forma apropiada, se obtiene un generador de período máximo.

Resultan idóneos para sistemas de simulación gracias a su alta eficiencia y buen comportamiento estadístico. Por eso, su uso e implementación está muy extendido en sistemas operativos, librerías y lenguajes. Sin embargo, no son

aptos para su uso en criptografía ya que son predecibles: sólo se requieren unos pocos valores contiguos de la secuencia para determinar las constantes a, b y m.

Los generadores congruenciales cuadráticos

$$X_n = (aX_{n-1}^2 + bX_{n-1} + c)\ mod\ m \quad \textbf{(1.2)}$$

o cúbicos

$$X_n = (aX_{n-1}^3 + bX_{n-1}^2 + cX_{n-1} + d)\ mod\ m \quad \textbf{(1.3)}$$

son, también, desaconsejables en criptografía dada su predecibilidad [29, 55, 78].

1.2.2.2 Registros de desplazamiento con retroalimentación

Las secuencias basadas en registros de desplazamiento retroalimentados son muy conocidas desde el inicio de la electrónica. Son muy utilizadas en distintas disciplinas por lo que existe abundante literatura acerca de este tema. Estos generadores están formados por dos partes: un registro de desplazamiento y una función de retroalimentación [68, 89, 93-94].

Un registro de desplazamiento retroalimentado es un registro de n bits, en el que se establece un orden secuencial de los bits, obteniéndose una estructura similar a una cola FIFO (primero en entrar, primero en salir); cada vez que se introduce un nuevo bit por la izquierda, los demás se desplazan una posición a la derecha, extrayéndose el bit menos significativo, que ocupa la posición más a la derecha.

La función de retroalimentación se aplica al estado actual del registro de desplazamiento para obtener el nuevo bit que se va a introducir; así, el bit que se introduce depende, de forma más o menos compleja, de los bits que contenga el registro de desplazamiento.

Los bits extraídos forman la secuencia conforme se va repitiendo el proceso de retroalimentación. La semilla la constituye el estado inicial del registro de desplazamiento. En la figura 1.2 se puede ver el esquema de una implementación de este tipo de generadores para un registro de 8 bits.

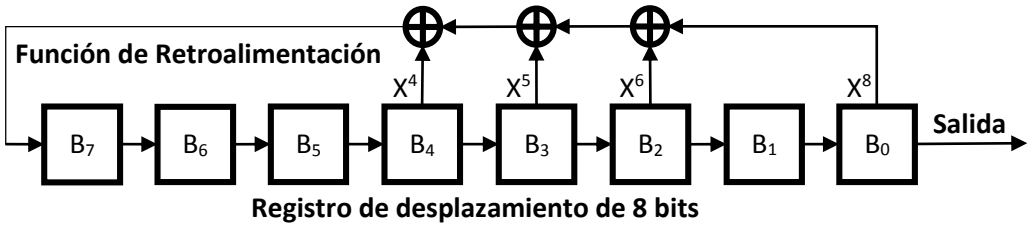


Figura 1.2: Forma de Fibonacci para un LFSR

Existen registros de desplazamiento con retroalimentación no lineal o NLFSR (*Non-Linear Feedback Shift Register*), pero su uso en criptografía es limitado.

Definición 1.26: LFSR (*Linear Feedback Shift Registers*)

Los registros de desplazamiento con retroalimentación lineal (LFSR) son aquellos que usan como bit de entrada una transformación lineal del registro.

La implementación habitual de la función de retroalimentación que proporciona la transformación lineal suele ser la operación $\oplus$ (XOR) entre ciertos bits del registro de desplazamiento, que se conoce como forma de Fibonacci (véase la figura 1.2). Esta forma de obtener el bit de retroalimentación resulta muy rápida y sencilla de implementar en hardware [20].

Un LFSR queda determinado por la longitud del registro n, el estado actual del registro y la lista de bits que se combinan mediante XOR. En teoría, un LFSR de longitud n puede pasar por $2^n - 1$ estados distintos, por lo que podría generar secuencias con un periodo máximo de $2^n - 1$. En la práctica, sólo ciertas combinaciones de bits a operar en la función de retroalimentación permiten que se alcance este máximo teórico. Para que un LFSR sea de período máximo, la configuración de su función de retroalimentación ha de cumplir que el polinomio formado por los bits elegidos más la constante 1 resulte un polinomio primitivo en $\mathbb{Z}_2$. El grado de este polinomio primitivo es la longitud del registro de desplazamiento y los exponentes de cada componente indican los bits que se operan con XOR para la retroalimentación.

Definición 1.27: Polinomio primitivo

Un polinomio primitivo es un polinomio mínimo (mónico e irreducible) de grado m y coeficientes en $\mathbb{Z}_p$ que genera todos los elementos (elemento primitivo) de la extensión del cuerpo finito $\mathrm{GF}(p^m)$.

El LFSR de la figura 1.2 tiene un período de $2^8 - 1$ ya que la configuración de su función de retroalimentación coincide con $x^8 + x^6 + x^5 + x^4 + 1$, que es polinomio primitivo en $\mathbb{Z}_2$.

Es posible obtener el mismo LFSR con la forma de Galois (véase la figura 1.3). Para una implementación software de un LFSR, la forma de Galois es más eficiente debido a que las operaciones XOR se pueden realizar de una sola vez sobre todo el registro y solamente es necesario examinar individualmente el bit de salida.

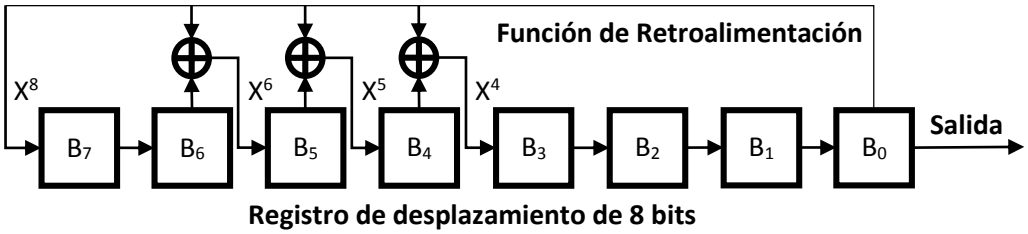


Figura 1.3: Forma de Galois para un LFSR

Nótese que con la forma de Galois, la interpretación que se hace de la configuración de la función de retroalimentación para su coincidencia con el polinomio asociado invierte el orden de los términos del polinomio respecto a los bits elegidos para operar.

Los LFSRs son buenos generadores pseudoaleatorios, pero tienen ciertas características que pueden ser problemáticas para su uso en criptografía. Para un LFSR de longitud n, el estado del registro son los n bits siguientes de la salida. Incluso si la función de retroalimentación es desconocida, se puede recuperar tras solo $2n$ bits de salida con el algoritmo Berlekamp-Massey, algoritmo que se usa para establecer la complejidad lineal, una de las métricas que se describen en la siguiente sección. Por ello, para aplicaciones criptográficas, no se recomienda el uso los LFSRs sin combinar su salida con alguna operación no lineal que evite que, conocido un tramo de texto plano y su correspondiente texto cifrado, un atacante pueda deducir la semilla y la función de retroalimentación asociada al LFSR. Se puede obtener más información sobre LFSR en [32, 43, 44, 61, 62, 77, 85-87, 89-90, 101].

Universitat d'Alacant
Universidad de Alicante

1.3 Medida de la aleatoriedad

1.3.1 Postulados de Golomb

Uno de los primeros intentos para establecer los criterios que ha de cumplir una secuencia aleatoria se deben a Solomon W. Golomb [44]. Para poder expresar correctamente sus postulados se consideran las siguientes definiciones sobre las secuencias binarias.

Definición 1.28: Secuencia N-periódica de periodo N y ciclo s^N

Sea $s = s_0, s_1, s_2, \ldots$ una secuencia infinita y $s^n = s_0, s_1, s_2, \ldots, s_{n-1}$ la subsecuencia de los n primeros términos de s. Se denomina n-periódica a la secuencia s si $s_i = s_{i+n}$ para todo $i \geq 0$. Esta secuencia es periódica si es

n-periódica para un entero positivo n cualquiera. El período de una secuencia periódica, s, es el menor entero positivo, N, para el que la secuencia es N-periódica. Si s es periódica con período N, entonces la subsecuencia s^N es el ciclo de s.

Definición 1.29: Racha, bloque y hueco

Dada una secuencia $s = s_0, s_1, s_2, \ldots$, una racha es una subsecuencia de s que consiste únicamente de bits 0 o únicamente de bits 1 consecutivos y no está precedida o seguida por el mismo bit. Un bloque es una racha de bits 1, mientras que un hueco es una racha de bits 0.

Definición 1.30: Función de autocorrelación

Sea $s = s_0, s_1, s_2, \ldots$ una secuencia periódica de período N. La función de autocorrelación es la función definida como

$$C(t) = \frac{1}{N}\sum_{i=0}^{N-1}(2s_i - 1)(2s_{i+t} - 1),$$

para $0 \leq t \leq N-1$.

La función de autocorrelación $C(t)$ mide la similitud entre s y su desplazada en t posiciones.

Una secuencia s, de período N, cumple los postulados de aleatoriedad de Golomb si:

I. En el ciclo s^N de s, el número de bits 1 difiere del número de bits 0 en no más de una unidad.

II. En el ciclo s^N de s, siendo r_i el número de rachas de longitud $i \geq 1$ y R el de rachas totales, entonces

$$\frac{r_i}{R} = \frac{1}{2^i}.$$

III. El valor absoluto de la función de autocorrelación es bivaluada, siendo K una constante:

$$|N\,C(t)| = \begin{cases} N, & \text{si } t = 0 \\ K, & \text{si } 1 \leq t \leq N-1 \end{cases}.$$

Definición 1.31: Pn-secuencia

Sea s una secuencia que cumple los postulados de aleatoriedad de Golomb, entonces s es una pn-secuencia o secuencia de pseudo ruido (*pseudo noise*).

Universitat d'Alacant
Universidad de Alicante

Los postulados de Golomb son condiciones necesarias pero no suficientes para que una secuencia sea aleatoria.

1.3.2 Métricas estadísticas

En la práctica, para considerar las secuencias de un generador como aleatorias se establece una serie de hipótesis estadísticas que comprobar.

Definición 1.32: Hipótesis estadística

Una hipótesis estadística, H_0, es una aserción acerca de la distribución de una o más variables aleatorias.

La comprobación consiste en tomar la salida del generador y aplicar una serie de test estadísticos [34, 68, 73].

Definición 1.33: Test estadístico

Un test estadístico es un procedimiento por el cual, a partir de los valores observados de las variables aleatorias, se llega a la aceptación o rechazo de una hipótesis estadística H_0.

Un test estadístico proporciona una medida de la fuerza de la evidencia proporcionada por los datos en contra de una hipótesis estadística, por lo que el resultado es probabilístico y no definitivo; por ello, pasar un test no implica que la secuencia sea realmente aleatoria, únicamente que el test no encuentra ningún fallo en esa secuencia. De esta forma, al pasar todos los test de forma satisfactoria, el generador es aceptado como aleatorio o, más correctamente, no rechazado.

Universidad de Alicante

Definición 1.34: Nivel de significación

El nivel de significación α del test de una hipótesis estadística H_0, es la probabilidad de rechazar dicha hipótesis aun cuando ésta sea cierta. Se define el valor de confianza como $1 - \alpha$.

Si el nivel de significación α es demasiado alto para un test entonces se corre el riesgo de rechazar secuencias que pertenezcan a un generador aleatorio correcto, si el nivel de significación es demasiado bajo se pueden aceptar secuencias que no sean aleatorias. En la práctica se emplea un rango para α entre 0.001 y 0.05, teniendo en cuenta que cuanto mayor sea α más restrictivo será el test.

A continuación se presentan cinco test estadísticos que se han usado en algunas de las baterías de experimentos de este trabajo. Algunos de estos test

siguen una distribución normal $N(0,1)$ y otros una distribución $\mathcal{X}^2$; de forma que los correspondientes valores de corrección usados se han extraído de tablas (véanse en [5, 68]) en que proporcionan el valor máximo que puede alcanzar el resultado de la prueba según el nivel de significación escogido para la distribución normal y el nivel de significación y grado de libertad en el caso de la distribución $\mathcal{X}^2$. En la definición de los test, n denota la longitud de la secuencia.

Definición 1.35: Test de frecuencia

El test de frecuencia o monobit consiste en comprobar que la frecuencia de bits con valor 1 y de bits con valor 0 sea, más o menos, la misma. Este test sigue una distribución $\mathcal{X}^2$ con 1 grado de libertad; n_0 y n_1 denotan el número de bits 0 y de bits 1 respectivamente:

$$X_1 = \frac{(n_0 - n_1)^2}{n}. \quad \textbf{(1.4)}$$

Definición 1.36: Test serial

El test serial o de parejas es una extensión del primero comprobando la frecuencia de las parejas de bits (00, 01, 10 y 11). Sigue una distribución $\mathcal{X}^2$ con 2 grados de libertad; n_{00}, n_{01}, n_{10} y n_{11} denotan el número de ocurrencias de las parejas de bits respectivas:

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}) - \frac{2}{n}(n_0^2 + n_1^2) + 1\ . \quad \textbf{(1.5)}$$

Definición 1.37: Test de póker

El test de póker comprueba que las secuencias de longitud m aparecen, razonablemente, el mismo número de veces; como se esperaría en una secuencia aleatoria. Para establecer la longitud m, se ha de tener en cuenta que m debe ser un entero positivo tal que $\left\lfloor \frac{n}{m} \right\rfloor \geq 5(2^m)$, k se define como $k = \left\lfloor \frac{n}{m} \right\rfloor$ y n_i denota el número de ocurrencias de secuencias de tipo i de longitud m con $1 \leq i \leq 2^m$. Este test sigue una distribución $\mathcal{X}^2$ con $2^m - 1$ grados de libertad:

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_1^2\right) - k\ . \quad \textbf{(1.6)}$$

Definición 1.38: Test de rachas

El test de rachas comprueba que el número de rachas de diferentes longitudes sea el esperado para una secuencia aleatoria. El número esperado de bloques de longitud i en una secuencia aleatoria es $e_i = \frac{(n-i+3)}{2^{i+2}}$ y debe coincidir

Universitat d'Alacant
Universidad de Alicante

con el de huecos de longitud i. Se define k como el mayor valor de i tal que $e_i \geq 5$. Se toman B_i y H_i como el número de ocurrencias de bloques y huecos, respectivamente, de longitud i, $1 \leq i \leq k$. El test de rachas sigue una distribución $\mathcal{X}^2$ con $2k-2$ grados de libertad:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(H_i - e_i)^2}{e_i}. \quad \textbf{(1.7)}$$

Definición 1.39: Test de autocorrelación

El test de autocorrelación consiste en comprobar las similitudes entre la secuencia y una copia (no cíclica) de la misma desplazada $d \leq \frac{n}{2}$ bits. Se toma $A(d)$ como el número de bits distintos entre ambas secuencias y el test sigue una distribución normal $N(0,1)$:

$$x_5 = \frac{2\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}. \quad \textbf{(1.8)}$$

Se puede encontrar más información acerca de los test y pruebas de secuencias aleatorias, así como de los conceptos estadísticos relacionados, en [23, 44, 48, 57, 63, 68, 97, 100].

1.3.3 Complejidad lineal y su perfil (Berlekamp-Massey)

La complejidad lineal es una métrica muy importante para el análisis de secuencias pseudoaleatorias, que permite detectar qué generadores son, en definitiva, equivalentes a un LFSR de longitud finita.

Definición 1.40: Complejidad lineal

La complejidad lineal de una secuencia binaria finita s^n, denotada por $L(s^n)$, se define como la longitud del LFSR más corto capaz de producir una secuencia con s^n como sus primeros n términos

La complejidad lineal esperada para una secuencia aleatoria de longitud suficiente está en torno a $\frac{n}{2}$, siendo n la longitud de la secuencia. Se espera, además, que la gráfica de la complejidad lineal respecto a la longitud de la secuencia (perfil de complejidad lineal) se ajuste a la gráfica de la recta $L(S^n) = \frac{n}{2}$.

Para el cálculo de la complejidad lineal se usa el algoritmo Berlekamp-Massey, que es de orden cuadrático y consiste en ir construyendo el LFSR bit a bit; comprobando discrepancias con la secuencia y modificando el LFSR construido convenientemente en función de dichas discrepancias.

El cálculo de la complejidad lineal es una buena herramienta, utilizada en este trabajo, para comprobar la adecuación criptográfica de secuencias pseudoaleatorias. Una buena complejidad lineal no implica que un generador sea seguro, pero una complejidad lineal baja sí implica que el generador es inseguro.

Más información sobre complejidad lineal y el algoritmo Berlekamp-Massey se puede encontrar en [24, 65, 68].

1.3.4 Análisis de aleatoriedad

Los test definidos en las secciones anteriores son sólo algunas de las pruebas de entre las múltiples hipótesis estadísticas que se pueden plantear para el estudio de secuencias pseudoaleatorias. Es bastante común agrupar estas pruebas en conjuntos o baterías: unas más generalistas, orientadas a descubrir ciertas debilidades en la supuesta aleatoriedad de las secuencias analizadas; y otras más específicas y adecuadas para el uso de generadores en aplicaciones criptográficas, que pueden necesitar cumplir unos requisitos más fuertes. Estas pruebas pueden ser útiles como un primer paso para determinar si un generador es adecuado para una aplicación criptográfica particular o no lo es. Sin embargo, ningún conjunto de pruebas estadísticas puede absolutamente certificar que un generador sea apropiado para el uso en una aplicación particular, es decir, la prueba estadística no puede servir como un sustituto del criptoanálisis. El diseño de estas pruebas y el criptoanálisis de generadores están fuera del alcance de este trabajo.

A continuación se van a referenciar dos análisis o conjuntos de pruebas que en la actualidad se consideran los más completos y exigentes. Por esta razón son los que se han utilizado para analizar las versiones finales del generador objeto de este trabajo.

1.3.4.1 PractRand

PractRand incluye una batería de test muy exhaustiva con algoritmos específicos y es más moderna que otras disponibles, permitiendo procesar secuencias de gran tamaño de forma muy eficiente y concurrente, pudiendo utilizar varios hilos de ejecución en paralelo. Su autor, Chris Doty-Humphrey, ha dejado bajo dominio público todo su trabajo, implementado en C++ y hospedado en *sourceforge* [41].

PractRand es una de las baterías de test de aleatoriedad más rápidas en la actualidad. Los test se aplican de forma sucesiva: si la secuencia resulta rechazada se entiende que no es aleatoria y el análisis se interrumpe; si por el contrario pasa el test se continúa el estudio con los siguientes test. El análisis

es incremental de forma que, conforme aumenta la longitud de la secuencia analizada, se aplican nuevos test, y se repiten otros con distintos parámetros o variables. Si el análisis no acaba por el rechazo en algún test, el fin del mismo vendrá dado por un límite de longitud de secuencia, que puede establecerse al lanzar el análisis, o porque el generador se interrumpa o deje de suministrar la secuencia. La rapidez del análisis permite analizar secuencias muy largas, del orden de Terabytes (incluso Petabytes).

1.3.4.2 TestU01

TestU01 es una serie de test estadísticos para generadores pseudoaleatorios similar a PractRand. Se trata de una herramienta más antigua, más lenta y de uso más complejo en la práctica; no obstante, está considerada como la más exigente en la actualidad. Incluye cinco conjuntos o baterías de test: alphabit, rabbit, small crush, crush y big crush. Es este último conjunto de 160 test (en su última versión 1.2.3) el más exigente de todos ellos.

En la práctica, TestU01 es una librería software; en este caso implementada en ANSI C, que incluye también algunos generadores pseudoaleatorios. La descripción de esta librería ha sido publicada en [58-59].

TestU01 analiza la secuencia con sus distintos test hasta completarlos todos. La longitud de la secuencia depende de los propios test de la batería escogida. El conjunto de test se completa a pesar de que alguno produzca un rechazo. Al terminar el análisis se proporciona un sumario donde se remarca qué test han resultado anómalos. El análisis se puede aplicar a distintas secuencias para determinar si las anomalías son comunes al generador. Dicho análisis es tan estricto que algunos autores consideran superado el análisis con varias de estas anomalías [74].

Universidad de Alicante

1.4 Cifrado en flujo

1.4.1 Características del cifrado en flujo y en bloque

Un cifrado en flujo es aquel que se realiza, incrementalmente, elemento a elemento (bits, dígitos, bytes, palabras, etc.). Lo más habitual es combinar el flujo de entrada de texto en claro con un flujo de elementos pseudoaleatorios (flujo clave); para combinar se suele utilizar la operación XOR y para los elementos bits o bytes.

En cualquier caso, el cifrado en flujo se realiza usando una función de transformación variable en el tiempo de manera que el cifrado de cada elemento es dependiente del estado actual en el que se encuentra esa función, por lo que también se conoce como sistema de cifrado de estado o con memoria [67, 84, 87, 91].

Definición 1.41: Cifrador en flujo

Un cifrador en flujo es un criptosistema que aplica una función de transformación variable a elementos individuales del texto en claro.

Un cifrado por bloques es el que opera sobre el texto en claro en grupos de elementos de longitud fija relativamente grandes (de 64, 128 o 256 bits, por ejemplo), llamados bloques, aplicándoles una función de transformación invariante, es decir, se aplica siempre la misma función; por lo que se dice que no poseen memoria, en referencia a que se guarde un estado del que dependa la trasformación, como sucede en un cifrado en flujo.

Definición 1.42: Cifrador en bloque

Un cifrador en bloque es un criptosistema que aplica una función de transformación fija a bloques del texto en claro.

El cifrado en bloque se realiza tomando un bloque de texto en claro como entrada y produciendo un bloque de texto cifrado [53]. El problema de cómo compatibilizar el tamaño de bloque con el tamaño del texto en claro se resuelve con un esquema de relleno y un modo de operación, que forman parte de la especificación completa del criptosistema. El esquema de relleno se refiere a cómo se debe rellenar el texto en claro para que su tamaño sea múltiplo del tamaño del bloque, mientras que el modo de operación determina cómo aplicar repetidamente una operación de cifrado a cantidades de datos de longitud mayor a la de un bloque.

Definición 1.43: Modo de operación

Un modo de operación criptográfico es el algoritmo que utiliza un cifrador en bloque para proveer servicios de seguridad sobre la información, como son la confidencialidad o la autenticidad.

El modo de operación influye en otros aspectos distintos a la seguridad como, por ejemplo, los relacionados con la accesibilidad (aleatoria o secuencial) o la implementación (parametrizable o no). Entre los modos de operación más conocidos están ECB, CBC, PCBC, CFB, OFB o CTR que son algoritmos bien documentados [33, 40, 50, 72, 79]. En la sección 1.5.1 se muestra uno de estos modos que permite usar un cifrador en bloque para cifrar en flujo; ya que, a

pesar de ser conceptualmente distintos, se puede implementar con un cifrador en bloque un cifrado en flujo.

Desde un enfoque funcional, los cifradores en flujo permiten cifrar comunicaciones bit a bit, lo que resulta ideal para sistemas hardware de tiempo real; mientras los cifradores en bloque son más cómodos para implementaciones en software, al trabajar con bloques grandes (palabras del procesador) y evitar la tediosa manipulación de bits de forma individual.

En teoría, los cifradores en flujo son más fáciles de analizar matemáticamente y existe un gran número de publicaciones al respecto, especialmente en el caso de los basados en LFSRs (véase, en la sección 1.2.2.2, la definición 1.26).

1.4.2 Cifrado de Vernam

El esquema más habitual de cifrado en flujo, es el de una transformación mediante la operación XOR del flujo de datos con un flujo clave. A este sencillo esquema se le denomina cifrado de Vernam, y al flujo clave, secuencia cifrante.

Definición 1.44: Secuencia cifrante

Se llama secuencia cifrante k_i a la secuencia de elementos que opera como flujo clave en un cifrado de Vernam.

Universitat d'Alacant
Universidad de Alicante

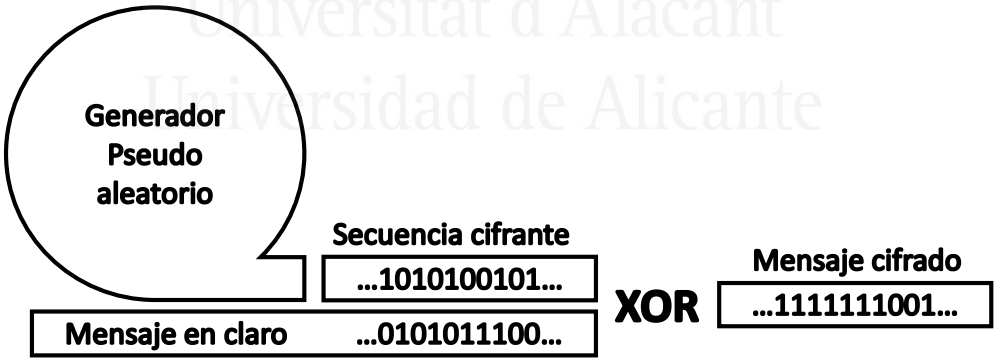


Figura 1.4: Cifrado de Vernam

Definición 1.45: Cifrado de Vernam

Siendo k_i la secuencia cifrante, m_i el texto en claro, c_i el texto cifrado, el cifrador de Vernam es aquel que combina durante el proceso de cifrado la secuencia cifrante con el texto en claro mediante la operación XOR binaria ($\oplus$):

$$c_i = m_i \oplus k_i \,, \tag{1.9}$$

y de igual forma durante el proceso de descifrado

$$m_i = c_i \oplus k_i \,, \tag{1.10}$$

ya que aplicando dos veces la misma operación XOR se obtiene el texto en claro original

$$m_i \oplus k_i \oplus k_i = m_i \,. \tag{1.11}$$

El esquema creado por Gilbert Vernam para usar en teletipos tenía inicialmente como flujo clave un bucle de cinta de papel. Posteriormente, en colaboración con Joseph Mauborgne, se sustituyó esa cinta por un sistema de claves aleatorias en libretas de un solo uso, cuyas siglas en inglés OTP (*One-Time Pad*) dan nombre a la secuencia cifrante ideal.

Definición 1.46: OTP (*One-Time Pad*)

Si una secuencia cifrante de un cifrador de Vernam es una secuencia aleatoria e independiente, se dice que es una OTP.

Claude Shannon demostró en su teoría de la información [91-92], que el cifrado de Vernam con OTP tiene la propiedad que denominó secreto perfecto; es decir, el texto cifrado no aporta absolutamente ninguna información adicional sobre el texto en claro. El inconveniente está en que la secuencia OTP ha de ser tan larga como el texto en claro, así que, el grave problema que supondría distribuir y gestionar las claves tan largas como el propio mensaje, se soluciona en la práctica utilizando un generador pseudoaleatorio para generar la secuencia cifrante. Véase la figura 1.4.

La salida de un generador pseudoaleatorio es una secuencia que, pese a ser completamente determinista y poderse reproducir convenientemente a la hora de descifrar, parece aleatoria para cualquier observador externo. Cuanto más se parezca la secuencia a una secuencia realmente aleatoria, más seguro será el cifrador. La semilla del generador puede interpretarse como clave del cifrador, ya que determina completamente la secuencia cifrante generada.

Así pues, el diseño de un cifrador en flujo que siga el esquema de Vernam, se reduce al diseño de un generador de secuencia cifrante. Éste consiste en un estado interno, una función de salida y una función de siguiente estado. El estado inicial del cifrador es la clave, o depende de ella mediante otra función. A partir del estado actual, se genera una salida determinada usando la función de salida. En cada iteración se obtiene un nuevo estado del cifrador aplicando la función de nuevo estado. Es decir,

$$e_o = f_k(K) \quad k_i = f_s(e_i) \quad e_{i+1} = f_e(e_i) \,, \tag{1.12}$$

donde e_i es el estado del cifrador en el instante i, K la clave, f_k la función de

estado inicial, f_e la función de nuevo estado y f_s la función de salida que proporciona la secuencia cifrante k_i.

1.4.3 Cifradores en flujo síncronos y autosincronizantes

Los cifradores en flujo que siguen el esquema de Vernam usando un generador de secuencia cifrante pertenecen a la clase de cifradores en flujo binarios aditivos [31].

Definición 1.47: Cifrador en flujo binario aditivo

Un cifrador en flujo binario aditivo es un cifrador en flujo síncrono donde el texto en claro, la secuencia cifrante y el texto cifrado son secuencias binarias y la secuencia cifrante y el texto en claro se combinan mediante la operación XOR entre bits.

En la definición de cifrador en flujo binario aditivo aparece un nuevo concepto, el de cifrador en flujo síncrono.

Definición 1.48: Cifrador en flujo síncrono

Un cifrador en flujo síncrono es aquel en que la secuencia cifrante se genera de forma independiente al texto en claro y el texto cifrado.

El proceso de cifrado de un cifrador en flujo síncrono se puede describir de la siguiente forma:

$$e_{i+1} = f_e(e_i, K) \quad k_i = f_s(e_i) \quad c_i = m_i \oplus k_i \ . \tag{1.13}$$

donde e_i es el estado interno en el instante i, K la clave, f_e la función de nuevo estado, f_s la función de salida, k_i la secuencia cifrante, m_i el mensaje en claro y c_i el cifrado.

Los generadores de secuencia cifrante de emisor y receptor han de estar sincronizados. Si uno de ellos se salta un ciclo o se pierde un bit del texto cifrado, entonces el resto del mensaje se descifrará de forma incorrecta. Este inconveniente se convierte en una ventaja frente a inserciones o borrados maliciosos del texto cifrado, ya que provocan una pérdida de sincronía que se detecta inmediatamente.

Lo habitual no suele ser la pérdida de bits, sino su alteración y, en este caso, los cifradores en flujo síncronos también tienen la ventaja de que ese tipo de errores afectan únicamente al bit alterado y no se propagan al resto de bits como le pasa a los cifradores en flujo autosincronizantes.

Universitat d'Alacant
Universidad de Alicante

Definición 1.49: Cifrador en flujo autosincronizante

Un cifrador en flujo autosincronizante es aquel en que la secuencia cifrante se genera como una función de la clave y de un número fijo de elementos del texto cifrado.

El proceso de cifrado de un cifrador en flujo autosincronizante se puede describir de la siguiente forma:

$$e_i = \left(c_{i-p}, c_{i-p+1}, c_{i-p+2}, \dots, c_{i-1}\right) \quad k_i = f_s(e_i, K) \quad c_i = m_i \oplus k_i \ . \qquad \textbf{(1.14)}$$

donde p es el número de elementos del cifrado c_i que constituyen el estado interno e_i en el instante i, K es la clave, f_s la función de salida, k_i la secuencia cifrante y m_i el mensaje en claro.

En este caso, como el estado interno depende de los p bits previos del texto cifrado, el descifrador se sincronizará automáticamente con el cifrador transcurridos p bits.

Este trabajo tiene como objeto un cifrador en flujo síncrono, para más información acerca de los cifradores en flujo autosincronizantes véanse [38, 45, 56, 88].

1.5 Generadores con los que se compara

En esta sección se realiza una breve descripción de los generadores pseudoaleatorios con los que se ha comparado el generador propuesto.

1.5.1 AES en modo OFB

El estándar de cifrado AES (Advanced Encryption Standard) es un sistema de cifrado por bloques diseñado para manejar longitudes de clave de 128, 192 y 256 bits con un tamaño de bloque de 128 bits. Realiza varias de sus operaciones internas a nivel de byte, interpretando estos como elementos de un campo de Galois $GF(2^8)$.

Tanto el método de cifrado como el de descifrado están estructurados en 'capas', todas ellas formadas por funciones reversibles. De esta manera, para descifrar basta con aplicar las funciones inversas de cada capa en orden contrario. Cada una de las capas de AES tiene una función específica y está diseñada para maximizar la no linealidad de las transformaciones, así como para impedir que la simetría del proceso de cifrado/descifrado conduzca a la

aparición de claves débiles. AES está basado en, lo que se conoce como, una red de sustitución-permutación; es rápido y relativamente fácil de implementar tanto en software como en hardware, requiriendo poca memoria.

Es conocido que cualquier cifrador en bloque, y en particular AES, utilizado en modo OFB (*output feedback*) [40, 50] se convierte en un generador de secuencia cifrante en el que, fijada una clave para AES, el vector de inicialización hace las veces de semilla (véase la figura 1.5).

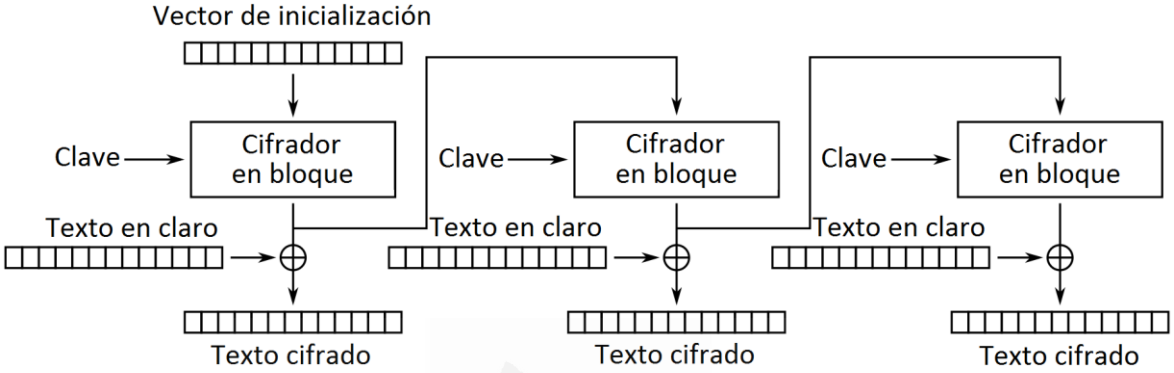


Figura 1.5: Cifrado en modo *output feedback* (OFB)

El estándar actual es en realidad una variante del original Rijndael que opera con bloques de 128 bits (4×4 bytes), y un tamaño de clave de 128, 192 o 256 bits (múltiplos de 32).

Se puede obtener más información sobre el algoritmo AES y los distintos modos de cifrado de los cifradores en bloque en [33, 39, 40, 50 , 72 , 89].

Universitat d'Alacant
Universidad de Alicante

1.5.2 Blum-Blum-Shub

Lenore Blum, Manuel Blum y Michael Shub propusieron el generador pseudoaleatorio Blum Blum Shub (BBS) a partir de la relación de recurrencia

$$X_{i+1} = X_i^2 \bmod n\,, \tag{1.15}$$

donde $n = pq$ con p y q primos congruentes $p, q \equiv 3 \bmod 4$.

El entero n se denomina entero de Blum y la semilla X_0 debe cumplir

$$X_0 = X^2 \bmod n\,, \tag{1.16}$$

donde X debe ser primo con n.

De cada X_i, se toma como salida sólo un bit (el de paridad o menos significativo, por ejemplo).

Tiene la característica de permitir el acceso aleatorio a un elemento i de la secuencia mediante la expresión

$$X_i = X_0^{2^i} \bmod \big((p-1)(q-1)\big)\,. \tag{1.17}$$

BBS ha quedado obsoleto por su lentitud, pero presenta unas buenas propiedades estadísticas que han servido de comparativa en este trabajo. Consúltense [4, 28, 30, 37, 47, 95] para más detalles acerca de BBS.

1.5.3 RC4

El algoritmo RC4 (Rivest *Cipher* 4, en honor a su creador Ron Rivest) es propiedad de RSA *Data Security*; y, aunque su código es en principio un secreto industrial desde su creación en 1987, se hizo público anónimamente en 1994, por lo que, para evitar problemas legales, en ocasiones se referencia como ARCFOUR, ARC4 o *Alleged*-RC4.

Se trata de un algoritmo sorprendentemente sencillo y eficaz; orientado a generar secuencias en unidades de un byte en las que los ciclos son bastante grandes, además de permitir claves de diferentes longitudes. Consiste en realidad en dos algoritmos bien diferenciados, que crean y emplean, respectivamente, una caja de sustitución de 8 × 8 (véase en la sección 3.4.1 la definición 3.21), es decir, un array de 256 bytes con una permutación de los números del 0 al 255. La caja de sustitución (*s-box*) se inicializa mediante el primer algoritmo, KSA (*Key Scheduling Algorithm*), que determina la permutación de los 256 elementos de la caja a partir de la clave. La permutación es consecuencia de una ronda de 256 intercambios de las componentes del array.

El segundo algoritmo, PRGA (*Pseudo-Random Generation Algorithm*), sigue el esquema de la figura 1.6, donde S es el array que implementa la *s-box*. En cada iteración el algoritmo determina qué dos componentes van a intercambiarse (de forma similar a como se hacía en KSA, pero sin implicar en este caso a la clave) para que la *s-box* evolucione. Las dos componentes elegidas para el intercambio (i y j) se suman (módulo 256) y determinan de qué componente se va a extraer el byte de salida B.

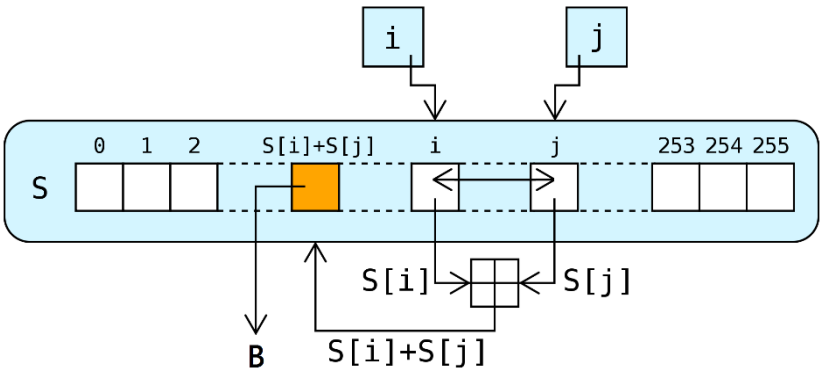


Figura 1.6: Esquema de funcionamiento del PRGA de RC4

En cada iteración, el índice i toma su nuevo valor de la expresión $(i+1) \bmod 256$, tanto en PRGA como en KSA, mientras que j lo hace de $(j+S_i) \bmod 256$ en PRGA y de $(j+S_i+K_i) \bmod 256$ en KSA, siendo K_i el byte i de la clave (vector de 256 bytes relleno a partir de la clave).

Que la caja de sustitución evolucione lentamente, realizándose un nuevo intercambio en cada iteración, complica su criptoanálisis. Su uso ha estado muy extendido en protocolos y aplicaciones de las últimas décadas (SSL, WEP, Oracle *Secure* SQL, etc.), y durante muchos años ha resultado inmune a los ataques a los que se ha sometido a los cifradores en flujo. Pero algunos problemas, bien relacionados con la implementación o con ciertos sesgos en la secuencia cifrante [52]; han hecho que se recomiende abandonar su uso en algunos protocolos. Desde febrero de 2015 el uso de RC4 está prohibido en TLS por la RFC 7465. Para más información sobre RC4 se puede consultar [70, 81, 83].

1.5.4 Salsa20

Salsa20, creado por Daniel J. Bernstein, es un cifrador en flujo basado en operaciones ARX (*add-rotate-xor*); es decir, suma modular (en este caso mod 2^{32}), rotación de bits y XOR.

El estado interno es una matriz de 4×4 palabras de 32 bits, que se inicializa con los 256 bits de la clave de donde se extraen 8 palabras. Las 8 palabras restantes para completar las 16 de la matriz se obtienen de la siguiente forma: 2 de la posición (un contador), 2 de un valor de un solo uso (*nonce*) y 4 que son valores fijos al algoritmo.

Universitat d'Alacant
Universidad de Alicante

Las operaciones ARX se encadenan en una única trasformación

$$R(a,b,c,k) = b \oplus \left((a \boxplus c) \lll k\right) \tag{1.18}$$

donde a, b y c son elementos de la matriz de estado, $\otimes$ es la operación XOR, $\boxplus$ la suma en aritmética modular y $\lll$ una rotación de k bits sobre un registro de 32 bits. La trasformación se aplica al propio elemento b que cambia su valor por el del resultado de $R(a,b,c,k)$.

Los elementos, para cada trasformación, se escogen alternando columnas impares, filas impares, columnas pares y filas pares, en cuatro cuartos de ronda, como se indica en la tabla 1.1, de valores para a, b, c y k. En cada ronda, a cada elemento se le aplican dos trasformaciones, una dependiente de los elementos de su fila y otra de los de su columna, por ello se dice que son rondas dobles.

Después de aplicar 20 rondas, la matriz de estado se toma como salida de la secuencia pseudoaleatoria, generando 32×16 bits (64 bytes). Luego se incrementa el contador y se repite el proceso para obtener el siguiente bloque.

El tamaño de registro del contador y la salida por iteración permiten secuencias de hasta 2^{70} bits de longitud.

Matriz de estado

e_0	e_1	e_2	e_3
e_4	e_5	e_6	e_7
e_8	e_9	e_{10}	e_{11}
e_{12}	e_{13}	e_{14}	e_{15}

1er ¼ de ronda

a	b	c	k
e_0	e_4	e_{12}	7
e_{10}	e_{14}	e_6	7
e_4	e_8	e_0	9
e_{14}	e_2	e_{10}	9
e_8	e_{12}	e_4	13
e_2	e_6	e_{14}	13
e_{12}	e_0	e_8	18
e_6	e_{10}	e_2	18

2º ¼ de ronda

a	b	c	k
e_0	e_1	e_3	7
e_{10}	e_{11}	e_9	7
e_1	e_2	e_0	9
e_{11}	e_8	e_{10}	9
e_2	e_3	e_1	13
e_8	e_9	e_{11}	13
e_3	e_0	e_2	18
e_9	e_{10}	e_8	18

3er ¼ de ronda

a	b	c	k
e_5	e_9	e_1	7
e_{15}	e_3	e_{11}	7
e_9	e_{13}	e_5	9
e_3	e_7	e_{15}	9
e_{13}	e_1	e_9	13
e_7	e_{11}	e_3	13
e_1	e_5	e_{13}	18
e_{11}	e_{15}	e_7	18

4º ¼ de ronda

a	b	c	k
e_5	e_6	e_4	7
e_{15}	e_{12}	e_{14}	7
e_6	e_7	e_5	9
e_{12}	e_{13}	e_{15}	9
e_7	e_4	e_6	13
e_{13}	e_{14}	e_{12}	13
e_4	e_5	e_7	18
e_{14}	e_{15}	e_{13}	18

Tabla 1.1: Tabla de trasformaciones en una ronda doble de Salsa20

Existen variantes de 8 y 12 rondas (Salsa20/8 y Salsa20/12). Salsa20/12 fue el más votado en la segunda fase de la competición eStream (*the ECRYPT Stream Cipher Project*) en el perfil software [22] y, finalmente, quedó como uno de los cuatro generadores propuestos para ese perfil.

Es posible consultar más información sobre Salsa20 en [21, 25-27, 82].

Universitat d'Alacant
Universidad de Alicante

1.5.5 HC128

El generador HC128 es otro de los seleccionados para el perfil software de eStream y debe su nombre a su autor Hongjun Wu (Hongjun*'s Cipher* 128 bits).

Como estado interno usa dos tablas (P y Q), cada una de 512 palabras de 32 bits, que actúan como *s-box* (cajas de sustitución, véase en la sección 3.4.1 la definición 3.21) y se inicializan mediante una clave y un vector de inicialización, ambos de 128 bits. Seis funciones definidas mediante operaciones de desplazamiento, rotación, XOR y aritmética modular, sirven para realizar los cambios de estado en las *s-box*, que evolucionan al ritmo de un elemento cambiado por iteración. Al mismo tiempo, por iteración, genera una salida de 32 bits como secuencia cifrante. Por cada par clave y vector de inicialización puede obtenerse una secuencia cifrante de 2^{64} bits.

El rendimiento de este algoritmo es muy bueno, pero resulta penalizado por una costosa inicialización de las tablas P y Q, por ello no se recomienda para aplicaciones que requieran una reinicialización frecuente.

Existe una versión de 256 bits (HC256) con el doble de entradas en las tablas P y Q que usa una clave y un vector de inicialización de 256 bits cada uno. Véanse [82, 98-99] para obtener más información sobre estos dos generadores.

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

2 Matrices triangulares superiores por bloques

2.1 Propiedades algebraicas

Como se ha indicado con anterioridad, en el capítulo 3 se analizará la aportación principal de este trabajo, consistente en un generador pseudoaleatorio binario basado en matrices triangulares superiores por bloques (TSB) con elementos en $\mathbb{Z}_p$, con p primo.

Dado p un número primo y $r, s \in \mathbb{N}$, se denota por $\mathrm{Mat}_r(\mathbb{Z}_p)$ y $\mathrm{Mat}_{r\times s}(\mathbb{Z}_p)$ a las matrices de tamaño $r \times r$ y $r \times s$, respectivamente, con elementos en $\mathbb{Z}_p$ y por $\mathrm{GL}_r(\mathbb{Z}_p)$ a las matrices invertibles de tamaño $r \times r$, también con elementos en $\mathbb{Z}_p$.

Se considera el conjunto Θ_n de matrices triangulares superiores por bloques con elementos en $\mathbb{Z}_p$ de la forma

$$M_n = \begin{bmatrix} A_1 & X_{1,2} & X_{1,3} & \cdots & X_{1,n-1} & X_{1,n} \\ 0 & A_2 & X_{2,3} & \cdots & X_{2,n-1} & X_{2,n} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & A_{n-1} & X_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & A_n \end{bmatrix}, \tag{2.19}$$

donde $A_k \in \mathrm{GL}_{r_k}(\mathbb{Z}_p)$ con $k = 1, 2, \dots, n$, $X_{i,j} \in \mathrm{Mat}_{r_i \times r_j}(\mathbb{Z}_p)$ con $i = 1, 2, \dots, n-1$ y $j = 2, 3, \dots, n$, y además 0 denota una matriz nula de tamaño apropiado.

Teorema 2.1

El conjunto Θ_n tiene estructura de grupo no abeliano para el producto de matrices.

Universitat d'Alacant
Universidad de Alicante

Demostración. Por definición de Θ_n, es evidente que la operación producto es cerrada.

La asociatividad es obvia por tratarse de matrices cuadradas.

El neutro para el producto viene dado por la matriz

$$I_{r_1+r_2+\cdots+r_n} = \begin{bmatrix} I_{r_1} & 0 & \cdots & 0 \\ 0 & I_{r_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_{r_n} \end{bmatrix}, \tag{2.20}$$

donde I_{r_k} es la matriz identidad en $\mathrm{GL}_{r_k}(\mathbb{Z}_p)$ para $k = 1, 2, \ldots, n$.

El inverso de un elemento cualquiera M_n es M_n^{-1} , matriz que existe por ser los bloques $A_k \in \mathrm{GL}_{r_k}(\mathbb{Z}_p)$, con $k = 1, 2, \ldots, n$.

Obsérvese, en la demostración del teorema anterior, que, M_n^{-1} se podría obtener con el método de Gauss; ahora bien, para este tipo particular de matrices se puede obtener como se indica a continuación.

Suponiendo que la matriz M_n^{-1} tiene la forma

$$M_n^{-1} = \begin{bmatrix} A_1^{-1} & X_{1,2}^{(-1)} & X_{1,3}^{(-1)} & \cdots & X_{1,n-1}^{(-1)} & X_{1,n}^{(-1)} \\ 0 & A_2^{-1} & X_{1,2}^{(-1)} & \cdots & X_{2,n-1}^{(-1)} & X_{2,n}^{(-1)} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & A_{n-1}^{-1} & X_{n-1,n}^{(-1)} \\ 0 & 0 & 0 & \cdots & 0 & A_n^{-1} \end{bmatrix}, \tag{2.21}$$

para cada $j = n, n-1, \ldots, 3, 2,$ los bloques $X_{i,j}^{(-1)}$ con $i = j-1, j-2, \ldots, 2, 1,$ se pueden obtener despejando $X_{i,j}^{(-1)}$en la expresión que se obtiene al multiplicar, por bloques, la fila i-ésima de M_n por la columna $j-$ é$sima$ de M_n^{-1} e igualar al correspondiente bloque nulo de la matriz identidad.

Por ejemplo, para $j = n$ e $i = n-1$, multiplicando la fila $(n-1)$-ésima de M_n por la columna $n-$ é$sima$ de M_n^{-1}, se tiene

$$A_{n-1}X_{n-1,n}^{(-1)} + X_{n-1,n}A_n^{-1} = 0\,; \tag{2.22}$$

de donde

$$X_{n-1,n}^{(-1)} = -A_{n-1}^{-1}X_{n-1,n}A_n^{-1}\,. \tag{2.23}$$

Para $j = n$ e $i = n-2$, multiplicando la fila $(n-2)$-ésima de M_n por la columna $n-$ é$sima$ de M_n^{-1}, se tiene

$$A_{n-2}X_{n-2,n}^{(-1)} + X_{n-2,n-1}X_{n-1,n}^{(-1)} + X_{n-2,n}A_n^{-1} = 0\,; \tag{2.24}$$

de donde

$$X_{n-2,n}^{(-1)} = -A_{n-2}^{-1}(X_{n-2,n-1}X_{n-1,n}^{(-1)} + X_{n-2,n}A_n^{-1}) , \tag{2.25}$$

esto es

$$X_{n-2,n}^{(-1)} = A_{n-2}^{-1}(X_{n-2,n-1}A_{n-1}^{-1}X_{n-1,n}A_n^{-1} - X_{n-2,n}A_n^{-1}) . \tag{2.26}$$

De igual modo se pueden obtener el resto de bloques.

Particular interés tienen los conjuntos Θ_2 y Θ_3 ya que en las propiedades del primero se apoyan los trabajos en los que se inspira el generador pseudoaleatorio propuesto y en las propiedades del segundo se apoya la propuesta aportada.

Para simplificar la notación, se considera que las matrices $M \in \Theta_2$ tienen la forma

$$M = \begin{bmatrix} A & X \\ 0 & B \end{bmatrix}, \tag{2.27}$$

con $\mathrm{A} \in \mathrm{GL}_r(\mathbb{Z}_p)$, $B \in \mathrm{GL}_s(\mathbb{Z}_p)$ y $X \in \mathrm{Mat}_{r\times s}(\mathbb{Z}_p)$. En este caso, el neutro es $I = \begin{bmatrix} I_r & 0 \\ 0 & I_s \end{bmatrix}$, donde I_r e I_s son las matrices identidad $r \times r$ y $s \times s$ en $\mathrm{GL}_r(\mathbb{Z}_p)$ y $\mathrm{GL}_s(\mathbb{Z}_p)$, respectivamente; y el inverso es $M^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}YB^{-1} \\ 0 & B^{-1} \end{bmatrix}$.

De análoga manera, para simplificar la notación, se considera que las matrices $M \in \Theta_3$ tienen la forma

$$M = \begin{bmatrix} A & Y & X \\ 0 & B & Z \\ 0 & 0 & C \end{bmatrix}, \tag{2.28}$$

con $\mathrm{A} \in \mathrm{GL}_r(\mathbb{Z}_p)$, $B \in \mathrm{GL}_s(\mathbb{Z}_p)$, $C \in \mathrm{GL}_t(\mathbb{Z}_p)$, $\mathrm{Y} \in \mathrm{Mat}_{r\times s}(\mathbb{Z}_p)$, $\mathrm{X} \in \mathrm{Mat}_{r\times t}(\mathbb{Z}_p)$ y $\mathrm{Z} \in \mathrm{Mat}_{s\times t}(\mathbb{Z}_p)$.

En este caso, el neutro viene dado por la matriz $I = \begin{bmatrix} I_r & 0 & 0 \\ 0 & I_s & 0 \\ 0 & 0 & I_t \end{bmatrix}$, donde I_r, I_s, I_t son las matrices identidad de tamaño $r \times r$, $s \times s$ y $t \times t$, respectivamente, y el inverso por

$$M^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}YB^{-1} & A^{-1}YB^{-1}Z\,C^{-1} - A^{-1}XC^{-1} \\ 0 & B^{-1} & -B^{-1}ZC^{-1} \\ 0 & 0 & C^{-1} \end{bmatrix}.$$

Universitat d'Alacant
Universidad de Alicante

2.2 Potencias de matrices TSB

El generador propuesto se obtiene a partir de las potencias de matrices del grupo Θ_3, partiendo de trabajos previos realizados por Álvarez et al. [1-3, 5-19, 96] para matrices del grupo Θ_2.

Para una matriz $M \in \Theta_2$ el siguiente teorema permite la obtención de las distintas potencias de forma simple.

Teorema 2.2

Sea una matriz $M = \begin{bmatrix} A & X \\ 0 & B \end{bmatrix} \in \Theta_2$ y $h \geq 0$ un entero no negativo.

Se tiene que

$$M^h = \begin{bmatrix} A^h & X^{(h)} \\ 0 & B^h \end{bmatrix}, \tag{2.29}$$

donde

$$X^{(h)} = \begin{cases} 0 & \text{si} \quad h = 0\,, \\ \sum_{i=1}^{h} A^{h-i} X B^{i-1} & \text{si} \quad h \geq 1\,. \end{cases} \tag{2.30}$$

Además, si $0 \leq t \leq h$ entonces

$$X^{(h)} = A^t X^{(h-t)} + X^{(t)} B^{h-t}\,, \tag{2.31}$$

$$X^{(h)} = A^{h-t} X^{(t)} + X^{(h-t)} B^t\,. \tag{2.32}$$

Demostración. Mediante inducción sobre h, se demostrará en primer lugar la expresión (2.29), que resulta obvia para $h = 0$ y $h = 1$.

Suponiendo cierta la expresión (2.29) para $h - 1$, se comprobará que lo sigue siendo para h.

$$M^h = MM^{h-1} = \begin{bmatrix} A & X \\ 0 & B \end{bmatrix} \begin{bmatrix} A^{h-1} & X^{(h-1)} \\ 0 & B^{h-1} \end{bmatrix} = \begin{bmatrix} A^h & X^{(h-1)} + XB^{h-1} \\ 0 & B^h \end{bmatrix}. \tag{2.33}$$

Por hipótesis de inducción, aplicando (2.30) se tiene

(2.34)

$$\begin{aligned} AX^{(h-1)} + XB^{h-1} &= A\sum_{i=1}^{h-1} A^{h-1-i}XB^{i-1} + XB^{h-1} \\ &= \sum_{i=1}^{h-1} A^{h-i}XB^{i-1} + XB^{h-1} \\ &= \sum_{i=1}^{h} A^{h-i}XB^{i-1} \\ &= X^{(h)}\,; \end{aligned}$$

obteniéndose la misma expresión que en (2.30).

También, si $0 \le t \le h$, se tiene

$$M^h = MM^{h-\mathrm{t}} = \begin{bmatrix} A^t & X^{(t)} \\ 0 & B^t \end{bmatrix}\begin{bmatrix} A^{h-\mathrm{t}} & X^{(h-\mathrm{t})} \\ 0 & B^{h-\mathrm{t}} \end{bmatrix} = \begin{bmatrix} A^h & A^tX^{(h-t)} + X^{(t)}B^{h-t} \\ 0 & B^h \end{bmatrix}. \quad \textbf{(2.35)}$$

Comparando este resultado con (2.29) se obtiene (2.31). De igual manera se demostraría (2.32).

Se ha extendido el teorema anterior para matrices $M \in \Theta_3$, obteniendo el siguiente teorema que permitirá la obtención de las potencias de este tipo de matrices de forma simple y eficiente.

Teorema 2.3

Dada

$$M = \begin{bmatrix} A & Y & X \\ 0 & B & Z \\ 0 & 0 & C \end{bmatrix} \in \Theta_3\,, \quad \textbf{(2.36)}$$

se consideran las diferentes potencias de M.

Tomando h como un entero no negativo se tiene que

$$M^h = \begin{bmatrix} A^h & Y^{(h)} & X^{(h)} \\ 0 & B^h & Z^{(h)} \\ 0 & 0 & C^h \end{bmatrix}, \quad \textbf{(2.37)}$$

donde

$$Y^{(h)} = \begin{cases} 0 & \text{si} \quad h = 0\,, \\ \displaystyle\sum_{i=1}^{h} A^{h-i}YB^{i-1} & \text{si} \quad h \ge 1\,, \end{cases} \quad \textbf{(2.38)}$$

Universitat d'Alacant
Universidad de Alicante

$$X^{(h)} = \begin{cases} 0 & \text{si} \quad h = 0\,, \\ \sum_{i=1}^{h} (A^{h-i} X C^{i-1} + \sum_{j=1}^{h-1} A^{h-i-j} Y B^{j-1} Z C^{i-1}) & \text{si} \quad h \geq 1\,, \end{cases} \tag{2.39}$$

$$Z^{(h)} = \begin{cases} 0 & \text{si} \quad h = 0\,, \\ \sum_{i=1}^{h} B^{h-i} Z C^{i-1} & \text{si} \quad h \geq 1\,. \end{cases} \tag{2.40}$$

Además, si $0 \leq k \leq h$, se tiene

$$Y^{(h)} = \mathrm{A}^{\mathrm{k}} Y^{(h-k)} + Y^{k} B^{h-k}\,, \tag{2.41}$$

$$X^{(h)} = \mathrm{A}^{\mathrm{k}} X^{(h-k)} + Y^{(k)} Z^{(h-k)} + X^{(\mathrm{k})} C^{h-k}\,, \tag{2.42}$$

$$Z^{(h)} = B^{k} Z^{(h-k)} + Z^{(\mathrm{k})} C^{h-k}\,. \tag{2.43}$$

Demostración. Se usará para la demostración el método de inducción sobre h.

Para h = 0 y h = 1, el resultado es obvio.

Se suponen ciertas las expresiones (2.38), (2.39) y (2.40) para $h-1$ y se demostrará que también lo son para h.

$$M^{h} = MM^{h-1} = \begin{bmatrix} A & Y & X \\ 0 & B & Z \\ 0 & 0 & C \end{bmatrix} \begin{bmatrix} A^{h-1} & Y^{(h-1)} & X^{(h-1)} \\ 0 & B^{h-1} & Z^{(h-1)} \\ 0 & 0 & C^{h-1} \end{bmatrix} = \begin{bmatrix} A^{h} & Y^{(h)} & X^{(h)} \\ 0 & B^{h} & Z^{(h)} \\ 0 & 0 & C^{h} \end{bmatrix}. \tag{2.44}$$

Por hipótesis de inducción, y aplicando (2.38), se tiene que

$$\begin{aligned} Y^{(h)} &= AY^{(h-1)} + YB^{h-1} \\ &= A \sum_{i=1}^{h-1} A^{h-i-1} Y B^{i-1} + YB^{h-1} \\ &= \sum_{i=1}^{h-1} A^{h-i} Y B^{i-1} + A^{0} Y B^{h-1} \\ &= \sum_{i=1}^{h} A^{h-i} Y B^{i-1}\,, \end{aligned} \tag{2.45}$$

de manera que la expresión (2.38) se verifica también para h.

Finalmente, para demostrar también las expresiones (2.39) y (2.40), se considera que

$$M = \begin{bmatrix} A & Y & X \\ 0 & B & Z \\ 0 & 0 & C \end{bmatrix} = \begin{bmatrix} A_1 & X_1 \\ 0 & C_1 \end{bmatrix} \tag{2.46}$$

Universitat d'Alacant
Universidad de Alicante

donde $A_1 = \begin{bmatrix} A & Y \\ 0 & B \end{bmatrix}$, $X_1 = \begin{bmatrix} X \\ Z \end{bmatrix}$ y $C_1 = C$.

Basándose en las expresiones (2.29) y (2.30) para matrices triangulares superiores con 2×2 bloques, para la matriz M considerada en (2.46),

$$M^h = MM^{h-1} = \begin{bmatrix} A_1 & X_1 \\ 0 & C_1 \end{bmatrix} = \begin{bmatrix} A_1^{h-1} & X_1^{(h-1)} \\ 0 & C_1^{h-1} \end{bmatrix} = \begin{bmatrix} A_1^{h} & X_1^{(h)} \\ 0 & C_1^{h} \end{bmatrix} \tag{2.47}$$

donde

$$\begin{aligned} X_1^{(h)} &= \sum_{i=1}^{h} A_1^{h-i} X_1 {C_1}^{i-1} \\ &= \sum_{i=1}^{h} \begin{bmatrix} A^{h-i} & Y^{(h-i)} \\ 0 & B^{h-i} \end{bmatrix} \begin{bmatrix} X \\ Z \end{bmatrix} C^{i-1} \\ &= \sum_{i=1}^{h} \begin{bmatrix} A^{h-i}X + Y^{(h-i)}Z \\ B^{h-i}Z \end{bmatrix} C^{i-1} \\ &= \sum_{i=1}^{h} \begin{bmatrix} A^{h-i}XC^{i-1} + Y^{(h-i)}ZC^{i-1} \\ B^{h-i}ZC^{i-1} \end{bmatrix} \\ &= \begin{bmatrix} X^{(h)} \\ Z^{(h)} \end{bmatrix} . \end{aligned} \tag{2.48}$$

En consecuencia,

$$X^{(h)} = \sum_{i=1}^{h} \left(A^{h-i}XC^{i-1} + Y^{(h-i)}ZC^{i-1} \right), \tag{2.49}$$

y, como

$$Y^{(h-i)} = \sum_{j=1}^{h-i} A^{h-i-j} Y B^{j-1} , \tag{2.50}$$

resulta que

$$X^{(h)} = \sum_{i=1}^{h} (A^{h-i}XC^{i-1} + \sum_{j=1}^{h-1} A^{h-i-j} Y B^{j-1} Z C^{i-1}) , \tag{2.51}$$

$$Z^{(h)} = \sum_{i=1}^{h} B^{h-i} Z C^{i-1} , \tag{2.52}$$

Universitat d'Alacant
Universidad de Alicante

y las expresiones (2.39) y (2.40) también son ciertas para h.

Si $0 \leq k \leq h$, se tiene

$$\begin{aligned} M^h &= M^k M^{h-k} \\ &= \begin{bmatrix} A^k & Y^{(k)} & X^{(k)} \\ 0 & B^k & Z^{(k)} \\ 0 & 0 & C^k \end{bmatrix} \begin{bmatrix} A^{h-k} & Y^{(h-k)} & X^{(h-k)} \\ 0 & B^{h-k} & Z^{(h-k)} \\ 0 & 0 & C^{h-k} \end{bmatrix} \\ &= \begin{bmatrix} A^h & A^k Y^{(h-k)} + Y^{(k)} B^{h-k} & A^k X^{(h-k)} + Y^{(k)} Z^{(h-k)} + X^{(k)} C^{h-k} \\ 0 & B^h & B^k Z^{(h-k)} + Z^{(k)} C^{h-k} \\ 0 & 0 & C^h \end{bmatrix}. \end{aligned} \tag{2.53}$$

Comparando este resultado con la expresión (2.37) se obtienen las expresiones (2.41), (2.42) y (2.43).

Como consecuencia, si a y b son dos enteros tales que $a + b \geq 0$, tenemos

$$Y^{(a+b)} = A^a Y^{(b)} + Y^{(a)} B^b \,, \tag{2.54}$$

$$X^{(a+b)} = A^a X^{(b)} + Y^{(a)} Z^{(b)} + X^{(a)} C^b \,, \tag{2.55}$$

$$Z^{(a+b)} = B^a Z^{(b)} + Z^{(a)} C^b \,. \tag{2.56}$$

En el caso $a = h - 1$ y $b = 1$, tenemos

$$Y^{(h)} = A^{h-1} Y + Y^{(h-1)} B \,, \tag{2.57}$$

$$X^{(h)} = A^{h-1} X + Y^{(h-1)} Z + X^{(h-1)} C \,, \tag{2.58}$$

$$Z^{(h)} = B^{h-1} Z + Z^{(h-1)} C \,, \tag{2.59}$$

que son las expresiones en las que se fundamenta el generador propuesto.

Universitat d'Alacant
Universidad de Alicante

2.3 Orden de las matrices TSB

Definición 2.1: Orden de una matriz

Se llama orden de una matriz $A \in \mathrm{GL}_r(\mathbb{Z}_p)$ al menor entero positivo m verificando $A^m = I_r$

Teorema 2.4

Dada una matriz $\mathrm{M} \in \Theta_n$ de orden m, el conjunto $\mathrm{H} = \{I_{r_1+r_2+\cdots+r_n}, M, M^2, M^3, \ldots, M^{m-2}, M^{m-1}\}$ es un subgrupo cíclico de Θ_n de orden m.

Demostración. Por construcción, M es un generador de H.

El neutro para el producto es $I_{r_1+r_2+\cdots+r_n}$ y el inverso de M^k es M^{m-k}, para $k = 1,2,\cdots,m-1$.

El generador pseudoaleatorio desarrollado hace uso de potencias de matrices del grupo Θ_3. Si elegimos una matriz $M \in \Theta_3$ de orden m, el conjunto $\mathrm{H} = \{I_{r+s+t}, M, M^2, M^3, \ldots, M^{m-2}, M^{m-1}\}$ determina la secuencia pseudoaleatoria ya que de cada uno de sus elementos se extraerán una serie de bits. Podemos afirmar, por tanto, que el periodo de la secuencia generada está vinculado al orden de la matriz $M \in \Theta_3$ elegida. Entre otras muchas propiedades, una secuencia pseudoaleatoria que vaya a ser utilizada como secuencia cifrante debe tener un periodo muy grande por lo que debemos garantizar que el orden de M también lo es.

Definición 2.2: Matriz asociada a un polinomio

Sea $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} + x^n$ un polinomio mónico en $\mathbb{Z}_p[x]$, su matriz asociada $n \times n$ tiene la forma

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}, \tag{2.60}$$

es conocida como matriz asociada a f (*Frobenius companion matrix*).

Si f es un polinomio irreducible, el orden de A es igual al orden de cualquier raíz de f (elementos de $\mathbb{F}_{p^n}$) y el orden de A divide $p^n - 1$ (véanse [61-62]). Además, en el caso de que f sea un polinomio primitivo en $\mathbb{Z}_p[x]$, el orden de A es exactamente $p^n - 1$. En consecuencia, trabajando con polinomios primitivos en $\mathbb{Z}_p[x]$ se pueden obtener matrices cuyo orden sea máximo.

Odoni, Varadharajan y Sanders [76] proponen un esquema extendido, basado en la construcción de matrices por bloques de la forma

$$\bar{M} = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & A_k \end{bmatrix}, \tag{2.61}$$

donde A_i es la matriz asociada a f_i, siendo f_i polinomios primitivos diferentes en $\mathbb{Z}_p[x]$ de grado n_i, para $i = 1, 2, \ldots, k$.

El orden de A_i, para $i = 1,2,\ldots,k$, es $p^{n_i} - 1$ y, por tanto, el de $\bar{M}$ es

$$mcm(p^{n_1} - 1, p^{n_2} - 1, \ldots, p^{n_k} - 1)\,. \tag{2.62}$$

En el generador desarrollado, se han construido matrices $M \in \Theta_3$ (considerando la notación de la expresión (2.28)) tomando los bloques A, B y C como matrices asociadas a polinomios primitivos en $\mathbb{Z}_p[x]$ de grado r, s y t, respectivamente, con los que se garantiza que su orden es, como mínimo,

$$mcm(p^r - 1, p^s - 1, p^t - 1)\,. \qquad \textbf{(2.63)}$$

La autocorrelación obtenida en el análisis empírico de las secuencias generadas (descrito en el capítulo 3) indica la no presencia de subperiodos, en concordancia con el hecho de que si el orden de M vale m se tiene que $M^m = I_{r+s+t}$ y, por tanto, $A^m = I_r$, $B^m = I_r$, $C^m = I_t$, $Y^{(m)} = 0_{r\times s}$, $X^{(m)} = 0_{r\times t}$ y $Z^{(m)} = 0_{s\times t}$. Si para algún valor $m_0 < m$ se anulasen los bloques superdiagonales ($Y^{(m_0)} = 0_{r\times s}$, $X^{(m_0)} = 0_{r\times t}$, $Z^{(m_0)} = 0_{s\times t}$), M^{m_0+1} no tendría necesariamente que ser igual a M, ya que los bloques diagonales de M^{m_0} ($A^{m_0}, B^{m_0}, C^{m_0}$) no pueden ser al mismo tiempo la identidad, por ir en contra de que el orden sea m. En consecuencia, el bit (o los bits) generado a partir de M^{m_0+1} no tiene por qué coincidir con el generado a partir de M. De igual manera, el bit (o los bits) generado a partir de M^{m_0+2} no tiene por que coincidir con el generado a partir de M^2 y así sucesivamente.

En el capítulo 3 (tabla 3.1) se mostrará que con valores pequeños de p, r, s, t, es posible conseguir órdenes muy grandes de M.

Universitat d'Alacant
Universidad de Alicante

3 Cifrado en flujo basado en matrices triangulares superiores por bloques

3.1 Generador pseudoaleatorio basado en matrices TSB de 3×3 bloques

En esta sección se describe el diseño del generador pseudoaleatorio propuesto que, básicamente, consiste en elegir una matriz $M \in \Theta_3$ de la forma (2.28) con un orden suficientemente grande, generar las sucesivas potencias de M e ir extrayendo bits de los bloques superiores en cada una de estas potencias.

El generador está determinado por el primo p y el tamaño de los bloques A, B, C (r, s, t, respectivamente), bloques que se obtienen como matrices asociadas a polinomios primitivos en $\mathbb{Z}_p[x]$ (de grados r, s, t, respectivamente) que garanticen un orden suficiente.

Los valores elegidos para p, r, s, t determinan también el tamaño de los bloques iniciales X, Y, Z, y sucesivos $X^{(h)}, Y^{(h)}, Z^{(h)}$, obtenidos conforme a las expresiones (2.36) y (2.37).

Los valores iniciales de los bloques X, Y, Z conforman la semilla de la secuencia de bits; la cual, a su vez, se obtendrá mediante algún sistema de extracción de la secuencia de bloques $X^{(h)}, Y^{(h)}, Z^{(h)}$ con $h = 2, 3, \ldots$.

Definición 3.1: Sistema de extracción

El sistema de extracción consiste en una serie de operaciones sobre los bloques superiores a la diagonal de una matriz TSB, con el fin de obtener una cantidad de bits determinada.

Universitat d'Alacant
Universidad de Alicante

En versiones previas de este tipo de generador, basado en matrices TSB de 2×2 bloques (Álvarez et al. [1-3, 5, 7, 9-17, 19, 96]), el sistema de extracción consistía en obtener un elemento en $\mathbb{Z}_p$ suma de todos los elementos del bloque $X^{(h)}$, del cual, a su vez, se extraía el bit menos significativo.

Definición 3.2: Bits menos significativos (*bms*)

Se entiende que los bits menos significativos de un entero, o *bms*, son los bits más a la derecha (bits con menor peso) de la expresión binaria de ese entero.

Reducir todos los elementos del bloque $X^{(h)}$ a un único elemento del que extraer el *bms*, produce un solo bit por cada iteración h.

Definición 3.3: Bits por iteración (*bpi*)

Se denomina bits por iteración, o *bpi*, al número de bits obtenido por el sistema de extracción en cada iteración del algoritmo.

El *bpi* del sistema de extracción influye en el rendimiento del generador. En el sistema de extracción comentado, por ejemplo, si se extrae más de un bit de la suma de elementos (2 o 4 *bms*, por ejemplo), el *bpi* aumenta sin coste computacional, incrementando así la velocidad de generación de bits. En contraposición, la secuencia generada puede obtener un resultado más deficiente en los test de aleatoriedad (véase sección 1.3).

Una forma de mejorar la secuencias producidas por el generador en términos estadísticos, consiste en realizar un proceso de filtrado previo a la salida definitiva.

Definición 3.4: Proceso de filtrado

Se llama proceso de filtrado al conjunto de transformaciones que se aplican a la secuencia de bits obtenida tras el sistema de extracción, proporcionando la secuencia definitiva.

El filtrado puede resultar conveniente, no solo con sistemas de transacción que aumenten el *bpi*, sino en cualquier caso, si la mejora en los test compensa su coste computacional.

En el generador basado en matrices triangulares de 2×2 bloques se experimentaba un fenómeno de sesgo (definición 1.17) reflejado principalmente en los test estadísticos de frecuencia y serial (definición 1.35 y definición 1.36). Dado el riesgo criptográfico que supone usar una secuencia cifrante con un sesgo elevado, puesto que vulnera el principio de impredecibilidad de una secuencia aleatoria, se aconsejaba aplicar el filtrado del bit de paridad a la secuencia.

Universitat d'Alacant
Universidad de Alicante

Este proceso de filtrado, basado en la operación binaria XOR, se ha incluido entre los probados con el nuevo generador basado en matrices TSB de 3×3 con el nombre de filtrado XOR.

Definición 3.5: Filtrado XOR

Es el proceso de filtrado consistente en operar bit a bit la secuencia mediante XOR (operador $\otimes$) con el bit anterior, de forma que

$$k_0 = 0 \quad k_i' = k_i \otimes k_{i-1} \quad i = 1, 2 \dots, n \tag{3.64}$$

donde k_i son los bits obtenidos del método de extracción correspondiente y k_i' los de la secuencia obtenida tras aplicar el filtro.

Esta tipo de filtrado puede ser adaptado para aplicarse de forma directa mediante su operación byte a byte.

Las baterías de experimentación realizadas en este trabajo han sido realizadas en distintas fases sucesivas, siendo el objetivo final de todas ellas obtener una parametrización adecuada que garantice las buenas propiedades criptográficas de las secuencias producidas. Esta parametrización comprende la elección del primo p y los tamaños de bloque r, s, t; así como del método de extracción y, en su caso, del proceso de filtrado adecuado, para que, conjuntamente a la combinación p, r, s, t escogida, el resultado sea el de un generador con unas buenas propiedades criptográficas.

Definición 3.6: Combinación

Se denota por combinación a la 4-tupla formada por un primo p y tres tamaños r, s, t correspondientes a los bloques A, B, C, respectivamente.

Universitat d'Alacant
Universidad de Alicante

A lo largo de las sucesivas fases de experimentación, se ha ido acotando el amplio campo de parametrización posible y ajustando los métodos de extracción y filtrado; también se ha optimizado el algoritmo en función de los parámetros elegidos y el coste computacional asociado.

3.1.1 Restricciones a los parámetros

El periodo de la secuencia se ha considerado acotado inferiormente por la expresión (2.63), dado que de cada potencia de M se extraen uno o más bits (véase la sección 2.3).

Definición 3.7: Mínimo de bits para el periodo

Se entiende por mínimo de bits para el periodo (mbp) de una secuencia, al menor número de dígitos binarios necesario para expresar ese periodo.

Se ha considerado, por tanto, en el caso de $M \in \Theta_3$ (véase (2.63)), que el mbp viene dado por la expresión

$$\log_2\left(mcm(p^r - 1, p^s - 1, p^t - 1)\right). \quad \textbf{(3.65)}$$

Durante todo el estudio de parametrización se han considerado apropiadas para ser empleadas como secuencias cifrantes, exclusivamente las secuencias con un bmp igual o superior a 128, descartado todas las que no cumplan la siguiente expresión

$$\log_2\left(mcm(p^r - 1, p^s - 1, p^t - 1)\right) \geq 128. \quad \textbf{(3.66)}$$

Ya que la semilla viene determinada por los valores iniciales de los bloques X, Y, Z, la elección de los parámetros p, r, s, t también va a influir en las propiedades de esta. El tamaño de los bloques X, Y, Z puede comprometer la seguridad criptográfica de las secuencias generadas si el conjunto de todas las posibles semillas no es lo suficientemente grande, permitiendo un ataque por fuerza bruta.

El espacio de semillas viene dado por el producto cartesiano $\{\mathrm{Mat}_{r\times t}(\mathbb{Z}_p) \times \mathrm{Mat}_{r\times s}(\mathbb{Z}_p) \times \mathrm{Mat}_{s\times t}(\mathbb{Z}_p)\}$, con cardinal $p^{(rt+rs+st)}$, que es una cota superior del orden de M, dato que

$$mcm(p^r - 1, p^s - 1, p^t - 1) \leq (p^r - 1)(p^s - 1)(p^t - 1) \leq p^{r+s+t} \leq p^{(rt+rs+st)}. \quad \textbf{(3.67)}$$

Definición 3.8: Mínimo de bits para la semilla

Se entiende por el mínimo de bits para la semilla (mbs) al menor número de dígitos binarios necesarios para almacenar o expresar los valores de la semilla, que viene dado por la expresión

$$\log_2 p^{(rt+rs+st)}. \quad \textbf{(3.68)}$$

El cardinal del espacio de semillas coincide con el número de posibles bloques $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, donde se aplica el método de extracción, de forma que el mbs coincide con el máximo de bits que el sistema de extracción dispone para operar en cada iteración.

Considerando la expresión (3.67) se puede afirmar que las combinaciones p, r, s, t con $mbp \geq 128$ proporcionan semillas con $mbs \geq 128$ y son, por ambas razones, admisibles.

Definición 3.9: Combinación admisible

Se considera combinación admisible únicamente a aquella combinación, p, r, s, t, que cumple con la restricción (3.66).

Si $X = 0$, $Y = 0$ y $Z = 0$, entonces $X^{(h)} = 0$, $Y^{(h)} = 0$ y $Z^{(h)} = 0$ $\forall h \in \mathbb{N}$, como fácilmente se puede deducir de las expresiones (2.57), (2.58) y (2.59); de ahí que la semilla formada por los bloques nulos X, Y, Z deba evitarse por generar una secuencia no aleatoria.

En el caso $Y = 0$ o $Z = 0$, es fácilmente deducible, atendiendo a las expresiones (2.57) y (2.59), que $Y^{(h)} = 0$ o $Z^{(h)} = 0$ $\forall h \in \mathbb{N}$. En estos casos sí se genera una secuencia con cierta variabilidad, pero también se desaconseja aceptar como válida una semilla en tales circunstancias, ya que, al permanecer los bloques $Y^{(h)}$ o $Z^{(h)}$ nulos, y dependiendo del sistema de extracción o filtrado, la secuencia puede verse gravemente sesgada en los test estadísticos, resultando criptográficamente insegura.

La condición de que $Y \neq 0$ y $Z \neq 0$ se tiene en cuenta a la hora de escoger o generar la semilla inicial, sobre todo cuando se genera automáticamente a partir de una clave, como se verá en la sección 3.4.

Para obtener los bloques superiores de la matriz M^h, en cada nueva iteración, el algoritmo generador usa las expresiones (2.57), (2.58) y (2.59) derivadas del teorema 2.3. Como se puede deducir de dichas expresiones, no es necesario completar el cálculo de M^h en cada iteración, ya que solo se debe calcular A^h, B^h, $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, bloques que se necesita guardar para la extracción de bits y para operar en la siguiente iteración. Las potencias del bloque C no se calculan.

Para reducir el coste en el cálculo de cada iteración, se asume la conveniencia de asignar al bloque C el mayor de los tres tamaños r, s y t. Además, resulta interesante que ninguno de los tres bloques de la diagonal tenga el mismo tamaño para maximizar, de esta forma, el periodo de la secuencia (véase expresión (2.63)). Por ello, los valores para r, s y t en las combinaciones los consideramos en orden creciente

$$r < s < t \tag{3.69}$$

Intuitivamente hay que suponer que cuanto más grandes sean los bloques y el primo escogido, mayor aleatoriedad contendrán los bloques de donde se extraen los bits, y, por lo tanto, los sistemas de extracción pueden obtener más *bpi* sin perjuicio en los resultados en los test de aleatoriedad, con la mejora de rendimiento que ello supone. Sin embargo, se tiene el inconveniente de que el coste computacional asociado al cálculo en cada iteración también crece al aumentar los valores de r, s y t, obligando a buscar unos valores que proporcionen un equilibrio entre coste y rendimiento por iteración, sin perjuicio en la calidad de la secuencia.

Para facilitar la elección de unos valores apropiados para p, r, s y t, se ha considerado la suma de los tamaños de los bloques no nulos de M (véase (3.70)),

como una medida relacionada simultáneamente con la aleatoriedad y el rendimiento; ya que incluye, no sólo el tamaño de los bloques superiores X, Y y Z, sino también el de los bloques de la diagonal A, B y C.

Definición 3.10: Tamaño de los bloques no nulos (tbn)

Se denomina tamaño de los bloques no nulos (tbn) de una matriz $M \in \Theta_3$ de la forma (2.28) a la suma de los tamaños de los bloques de la diagonal, A, B, C, y de los bloques superiores, X, Y, Z, esto es

$$tbn(M) = r^2 + s^2 + t^2 + rt + rs + st \tag{3.70}$$

Recordemos que, como se indica en la sección 2.3, es necesario construir los bloques de la diagonal mediante polinomios primitivos. De forma que si

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{r-1} x^{r-1} + x^r \,, \tag{3.71}$$

$$g(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{s-1} x^{s-1} + x^s \tag{3.72}$$

$$h(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{t-1} x^{t-1} + x^t \tag{3.73}$$

son polinomios primitivos en $\mathbb{Z}_p[x]$ de grado r, s y t, respectivamente, entonces

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & & \\ 0 & 0 & 1 & \cdots & & \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{r-2} & -a_{r-1} \end{bmatrix}, \tag{3.74}$$

$$B = \begin{bmatrix} 0 & 1 & 0 & \cdots & & \\ 0 & 0 & 1 & \cdots & & \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -b & -b_1 & -b_2 & \ldots & -b_{s-2} & -b_{s-1} \end{bmatrix}, \tag{3.75}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & \cdots & & \\ 0 & 0 & 1 & \cdots & & \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -c_0 & -c_1 & -c_2 & \ldots & -c_{t-2} & -c_{t-1} \end{bmatrix}. \tag{3.76}$$

Conforme aumenta el grado de los polinomios primitivos, y sobre todo, el valor del primo p, la búsqueda de los mismos resulta cada vez más compleja, ya sea en bibliografías, bases de datos o mediante métodos de cálculo; hecho que ha supuesto una restricción más a la hora de escoger la parametrización del generador.

El algoritmo inicial del generador incorpora a su código una tabla de polinomios primitivos de donde extraer los polinomios correspondientes a los parámetros escogidos.

Universitat d'Alacant
Universidad de Alicante

En el apéndice C se muestra una colección de polinomios que se ha sido empleada para la confección de la tabla incorporada al algoritmo. Contiene más polinomios de los que son estrictamente necesarios para las baterías de pruebas.

Para las primeras tandas de experimentos, casi la totalidad de los polinomios usados han sido obtenidos de PIPS (Primitive and Irreducible Polynomial Server) [35], servidor web implementado como ejemplo de uso de la librería ZEN [36] (una herramienta para el cálculo rápido en extensiones y anillos finitos). Este servidor admite búsquedas de polinomios primitivos posibilidad de solicitar, en concreto, un trinomio.

Para simplificar la búsqueda y el almacenamiento en la tabla, se ha procurado obtener preferentemente un trinomio por cada primo y grado. En algunos casos, no se ha podido encontrar un trinomio y, en otros (los menos), ni siguiera un polinomio de más términos (al comienzo del apéndice C se concretan estas excepciones).

Al extender el generador a matrices TSB de 3×3 bloques se han tenido en cuenta bloques de tamaño 1×1, lo que resulta novedoso respecto a versiones anteriores de generadores basados en matrices TSB.

Para la construcción de estos bloques 1×1 se ha considerado que el único elemento de la matriz asociada a un polinomio primitivo de grado 1 es el opuesto en $\mathbb{Z}_p$ del término independiente del polinomio. Al tratarse de un polinomio mónico, el polinomio primitivo de grado 1 queda determinado solo por el valor de su término independiente, cuyo opuesto en $\mathbb{Z}_p$ es en realidad una raíz primitiva de ese conjunto, es decir, un generador de todo el conjunto $\mathbb{Z}_p$ mediante sucesivas potencias. Así pues, otra forma de ver la construcción de un bloque 1×1, es considerar que se construye tomando como su único elemento, una raíz primitiva.

No suele ser común considerar los polinomios primitivos de grado 1 en librerías de cálculo o las habituales tablas, por lo que en realidad se ha utilizado una tabla de raíces primitivas (también disponible en el apéndice C) para la construcción de los bloques de tamaño 1×1.

3.2 Primera aproximación

En una primera batería de experimentos, la búsqueda de parámetros adecuados se ha limitado a los 64 primos existentes entre los enteros del 3 al 313, y a tamaños de bloque menores o iguales a 7×7. No se han podido probar las combinaciones que incluyeran el tamaño 7 en los primos 173, 271 y 293, ni

el 6 para el primo 257, ya que no se disponía de polinomio primitivo para esos casos.

Son dos los sistemas de extracción utilizados en esta primera aproximación. El primero de ellos es totalmente equivalente al usado inicialmente por Álvarez [5], en el generador matricial TSB de 2×2 bloques, el cual ha sido comentado al principio de la sección 3.1. En dicha implementación se formulaba como una suma, mientras que en este caso se plantea como una operación XOR con todos los bits menos significativos de cada elemento de $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$. Al extraer un único bit por iteración del generador, el rendimiento de este sistema es de sólo 1 *bpi*; de ahí que haya sido etiquetado como sistema de extracción *slow*.

Definición 3.11: Extracción *slow*

Se denomina sistema de extracción *slow* aquel en el que se extrae un solo *bpi* realizando XOR del *bms* de todos los elementos de los bloques $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, correspondientes a cada iteración h.

El segundo sistema de extracción, denominado *fast*, viene motivado por la necesidad de aumentar el rendimiento; ya se proponía, también, como mejora al bajo rendimiento en el caso del generador de 2×2 bloques. Su implementación en este caso consiste en extraer el bit menos significativo de cada uno de los elementos de los bloques $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$ directamente como parte de la secuencia. El sistema extrae tantos bits por iteración como elementos tienen esos bloques, esto es

$$rt + rs + \mathrm{st}. \tag{3.77}$$

Definición 3.12: Extracción *fast*

Se denomina sistema de extracción *fast* a aquel en el que se extraen $rt + rs + \mathrm{st}$ *bpi*, eligiendo el *bms* de cada elemento de los bloques $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, correspondientes a cada iteración h.

En este segundo sistema de extracción, la combinación escogida influye de forma notable en el rendimiento del algoritmo.

La versión del algoritmo implementada para esta batería de experimentos admite escoger la combinación p, r, s, t, el tipo de extracción, *slow* o *fast*, la posibilidad de filtrado, XOR o no (véase definición 3.5 en la sección 3.1), y la longitud de la secuencia.

Mediante otro argumento, un identificador de tipo entero, se escoge una semilla; esta semilla es siempre la misma para el mismo identificador. De esta

Universitat d'Alacant
Universidad de Alicante

forma, se pueden comparar las secuencias generadas por la misma semilla con distintas opciones (sistema de extracción y/o filtrado).

El subconjunto de combinaciones admisibles dentro del conjunto de combinaciones posibles a probar en esta primera batería no es muy grande, según las restricciones planteadas en la sección 3.1 respecto al periodo y tamaño de la semilla; y se reduce a algunas combinaciones con primos mayores que 251. Por ello, y debido a que en esta primera aproximación se ha buscado únicamente estudiar el comportamiento en los test de aleatoriedad y la capacidad de producción de distintas combinaciones, se ha relajado la restricción al periodo y exigido únicamente un tamaño mínimo a la semilla ($mbs \geq 128$).

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
3	4	5	7	23	131	173
3	5	6	7	26	169	217

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
5	2	5	7	28	136	137
5	2	6	7	28	157	157
5	3	4	7	28	141	135
5	3	5	6	23	146	133
5	3	5	7	30	164	154
5	3	6	7	28	188	175
5	4	5	6	28	171	151
5	4	5	7	33	192	173
5	4	6	7	32	218	195
5	5	6	7	37	248	217

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
7	1	5	7	31	131	122
7	1	6	7	33	154	141
7	2	5	6	28	145	117
7	2	5	7	34	165	137
7	2	6	7	33	190	157
7	3	4	5	28	131	97
7	3	4	6	22	151	115
7	3	4	7	34	171	135
7	3	5	6	28	176	133
7	3	5	7	36	199	154
7	3	6	7	33	227	175
7	4	5	6	33	207	151
7	4	5	7	39	233	173
7	4	6	7	39	263	195
7	5	6	7	45	300	217

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
11	1	4	7	34	134	105
11	1	5	6	34	141	103
11	1	5	7	38	162	122
11	2	3	7	34	141	103
11	2	4	5	27	131	83
11	2	4	6	27	152	100
11	2	4	7	34	172	119
11	2	5	6	34	179	117
11	2	5	7	41	204	137
11	3	4	5	34	162	97
11	3	4	6	27	186	115
11	3	4	7	41	211	135
11	3	5	6	34	217	133
11	3	5	7	45	245	154
11	4	5	6	41	255	151
11	4	5	7	48	287	173
11	4	6	7	48	325	195
11	5	6	7	55	370	217

Tabla 3.1: Ejemplo de combinaciones seleccionables

En la tabla 3.1 se muestran las combinaciones, para tamaños de bloque inferiores o iguales a 7×7 y primos comprendidos entre 3 y 11, que se han considerado seleccionables exigiendo la restricción indicada en el párrafo anterior. Para cada combinación se indican el periodo de la secuencia y el tamaño de la semilla en mínimo de bits necesarios para ser expresados (columnas *mbp* y *mbs*), además del tamaño de los bloques no nulos (columna *tbn*).

Las filas para un mismo primo p vienen ordenadas por la combinación de r, s y t de menor a mayor, de forma que la primera combinación es la de los

bloques A, B y C de menor tamaño posible dentro de las combinaciones seleccionables (mbs mayores o iguales que 128).

Definición 3.13: Combinación *bottom*

Para un primo p, se denomina combinación *bottom* a la combinación p, r, s, t con los menores valores posibles de r, s y t y mbs mayor o igual que 128.

En la tabla 3.1, la última de todas las filas que empiezan por el mismo primo proporciona la combinación con mayores tamaños que se ha probado.

Definición 3.14: Combinación *top*

Para un primo p, la combinación, p, r, s, t, con los mayores valores de r, s y t, es la que se denomina combinación *top*.

En esta primera tanda de experimentos, la combinación *top* viene delimitada por el tamaño de bloque 7×7 que se ha fijado como tamaño máximo en esta batería de pruebas. La combinación *top* coincide siembre con la de mayor tamaño de los bloques no nulos (tbn (3.70)).

No sucede así con las combinaciones *bottom* en las que, sobre todo para los primos menores, no siempre coinciden con las de menor tbn. Por ello, se ha establecido una tercera categoría de combinaciones a la que se denomina *min*.

Definición 3.15: Combinación *min*

Para un primo p, se denomina combinación *min* a la combinación, p, r, s, t, con menor tbn y mbs mayor o igual a 128.

Universitat d'Alacant
Universidad de Alicante

top

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
3	5	6	7	26	169	217
5	5	6	7	37	248	217
7	5	6	7	45	300	217
11	5	6	7	55	370	217
...						
313	5	6	7	132	887	217

min

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
3	4	5	7	23	131	173
5	3	5	6	23	146	133
7	3	4	5	28	131	97
11	2	4	5	27	131	83
...						
313	1	3	4	49	157	45

bottom

p	*r*	*s*	*t*	*mbp*	*mbs*	*tbn*
3	4	5	7	23	131	173
5	2	5	7	28	136	137
7	1	5	7	31	131	122
11	1	4	7	34	134	105
...						
313	1	2	5	49	140	47

Tabla 3.2: Combinaciones *top*, *min* y *bottom*

Son muchas las combinaciones de tamaños de bloque inferiores o iguales a 7×7 seleccionables para un mismo primo, por lo que la batería de experimentos se ha limitado a probar sólo dos o tres combinaciones para cada primo. En concreto, se han probado las combinaciones *top*, *min* y *bottom*.

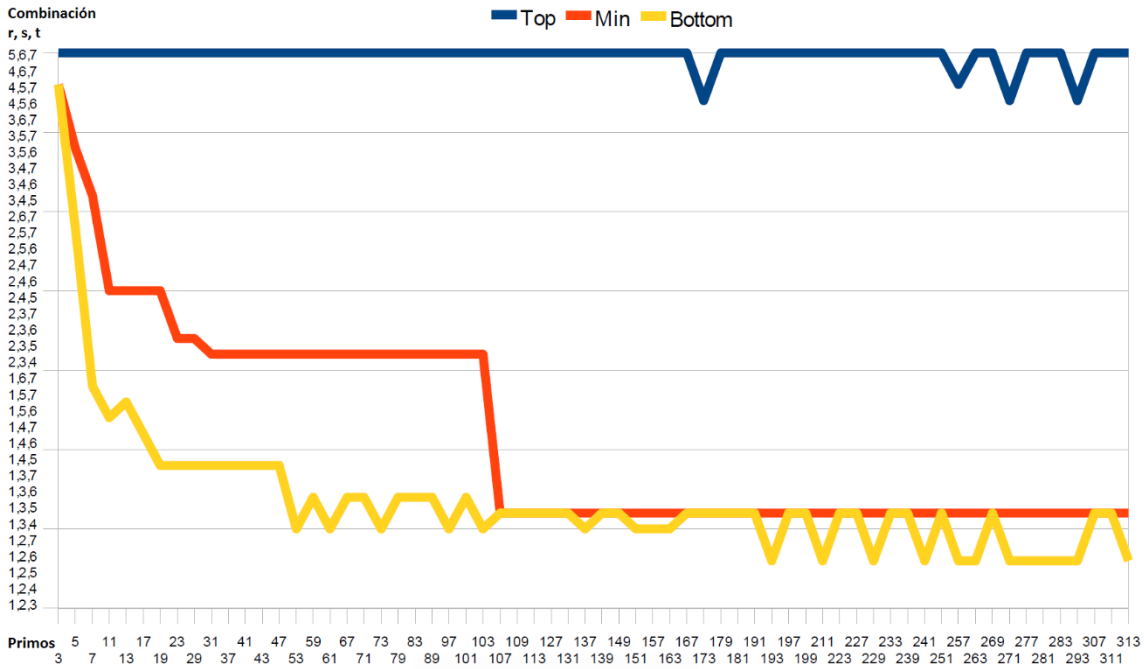


Figura 3.1: Combinaciones *top*, *min* y *bottom*

En la tabla 3.2 se indican las combinaciones probadas (*top*, *min* y *bottom*) para algunos primos. Para comprobar qué combinaciones se han escogido para los 64 primos es preferible consultar el apéndice A.1 o el gráfico de la figura 3.1, donde se muestra cuáles han sido las combinaciones seleccionadas. En dicha figura están representadas, en el eje de ordenadas, todas las combinaciones r, s y t en orden creciente de tamaños de bloque y, en el eje de abscisas, los primos del 3 a 313.

La línea *bottom* muestra la menor de las combinaciones posibles (menores tamaños r, s y t) con $mbs \geq 128$ para cada primo, mientras que la línea *min* marca, dentro de las combinaciones con $mbs \geq 128$, las combinaciones con menos elementos no nulos en la matriz M.

Se puede apreciar, claramente, cómo no coinciden las combinaciones *min* y *bottom* para todos los primos. La línea *top* debiera ser una recta horizontal correspondiente a la combinación 5, 6 y 7, la de mayores tamaños, salvo para los casos en los que no se han encontrado polinomios primitivos, lo que se refleja como picos invertidos en los primos 173, 257, 271 y 293.

Se han analizado 7.680 secuencias de 20.000 bits cada una. Estas 7.680 secuencias son el resultado de probar para cada uno de los 64 primos, 3 combinaciones de tamaños de bloque (*min*, *top* y *bottom*, a pesar de que la combinación *min* y *bottom* coinciden en algunos primos) y, para cada una de las tres combinaciones por primo, extraer la secuencia mediante el sistema de extracción *fast* y *slow*, así como con y sin filtrado. A su vez, para cada una de estas cuatro variantes (*fast* o *slow* por con o sin filtro), se han obtenido 10

secuencias con 10 semillas distintas (las semillas correspondientes a los identificadores del 1 al 10). En definitiva, se puede obtener el número total de secuencias como $64 \times 3 \times 2 \times 2 \times 10 = 7.680$.

La batería puede repetirse de nuevo, generando las mismas secuencias, si se usan los mimos identificadores de semillas. Se ha comprobado que ninguno de los diez identificadores genera una semilla con bloques Y o Z nulos.

Las secuencias han sido analizadas con los test propuestos en las secciones 1.3.2 y 1.3.3 para dos niveles de significación 0,001 y 0,1 (definición 1.34), comparándose, además, los resultados obtenidos con los de otros generadores pseudoaleatorios de referencia, BBS, AES y RC4 (véase sección 1.5).

El objetivo de esta primera batería tan exhaustiva de experimentos es acotar y dirigir la búsqueda de parámetros, permitiendo resolver cuestiones como ¿a partir de qué primo se obtienen resultados estadísticamente aceptables?, ¿qué tipo de combinaciones resultan más interesantes respecto a la aleatoriedad y coste computacional?, ¿qué combinaciones se ven más afectadas por el sesgo o el filtrado?, etc.

3.2.1 Análisis de los test

Los resultados obtenidos en los test correspondientes a esta primera batería de experimentos pueden ser consultados en el apéndice A.1.

En el test de frecuencia, sólo el uso de la extracción *fast* sin filtrado muestra irregularidades estadísticas; similares, además, en los tres tipos de combinaciones (*top*, *min* y *bottom*). Estas irregularidades desaparecen conforme aumenta el valor del primo.

En el test de pares o serial, la extracción *fast* se ve afectada de nuevo en primos bajos y por igual en los tres tipos de combinaciones sólo que, esta vez, el filtrado no consigue mejorar los resultados y las secuencias extraídas con *fast* y filtradas se ven igualmente afectadas. Igualmente, la solución consiste en aumentar el valor del primo, eliminándose el problema en ambas estrategias de extracción a partir de la mitad de la tabla.

En el caso de los test de póker, vuelve a surgir el mismo problema con la estrategia de extracción *fast,* tanto filtrada como sin filtrar. No obstante, en esta ocasión, los valores se corrigen rápidamente al aumentar el primo; siendo este fenómeno todavía más rápido en el test de póker 16.

Con el test de rachas vuelven a repetirse los resultados anómalos para las versiones *fast* (que resultan estadísticamente más deficientes), necesitando

primos algo mayores que en los test de póker para proporcionar secuencias estadísticamente aceptables.

El test de autocorrelación da resultados muy buenos en todos los tipos de combinación (*bottom*, *top*, y *min*). Sólo el sistema de extracción *fast* sin filtrado obtiene resultados no aceptables, aunque únicamente en los tres primeros primos.

En la prueba de *complejidad lineal* se obtienen resultados prácticamente perfectos. Como es esperable, se necesitaría un LFSR de tamaño próximo a 10000 para generar cualquiera de las secuencias de 20000 bit analizadas.

En general, el sistema de extracción *fast* resulta ser más conflictivo que el *slow* en cualquier combinación. A pesar de ello proporciona buenos resultados a partir de la segunda mitad de la tabla de primos (o antes atendiendo al nivel de significación más laxo).

3.2.2 Rendimiento

Se ha medido el tiempo, en ciclos de procesador, correspondiente a cada uno de los experimentos de la batería, con el objetivo de establecer el rendimiento asociado a cada combinación y sistema de extracción con o sin filtrado.

Los resultados se han comparado entre sí, y no con otros generadores pseudoaleatorios, por considerar que estas primeras versiones del generador distan mucho de resultar competitivas en términos de rendimiento frente a otros generadores. La idea inicial para esta primera batería de experimentos es la de averiguar sobre qué paramentos incidir para optimizar el rendimiento manteniendo resultados estadísticos adecuados.

Algunos resultados coinciden con los intuitivamente esperados: por ejemplo, el método de extracción *fast* resulta muchísimo más ventajoso en cuanto al coste computacional que el *slow*; así como que el coste computacional es mayor en las combinaciones *top* que en las *min* y *bottom*. Esto se aprecia con más claridad en el método de extracción *slow*; sin embargo, para el método *fast* la variación entre unas combinaciones y otras es mucho menor, sobre todo al aumentar el tamaño del primo.

El filtrado de la secuencia no supone un coste computacional significativo en comparación con la variación de los otros parámetros; así mismo, el tamaño del primo usado tampoco impacta el rendimiento de forma notable a partir de ciertos tamaños.

El método de extracción *fast* es, con mucha diferencia, el más ventajoso en términos de rendimiento de los dos estudiados; aun así, no resulta verdaderamente competitivo frente a otros algoritmos disponibles en la actualidad.

Como contrapartida, el método *fast* es el más afectado por el sesgo: necesita de primos más elevados y de la aplicación de un método de filtrado, como se ha comprobado en esta tanda de experimentos.

El método *fast* se beneficia de las combinaciones de bloques que proporcionan un alto *bpi*, aquellas que tiene muchos elementos en los bloques X, Y y Z, compensando con su extracción el coste de su cálculo.

Por todo ello resulta necesario centrar la búsqueda de parámetros en combinaciones que incluyan bloques de mayor tamaño que mejoren el coste computacional. También resulta interesante analizar nuevos primos más grandes, al apreciarse una clara mejoría de las características de aleatoriedad al aumentar dicho parámetro.

Cabe destacar el papel de las combinaciones con el tamaño del bloque A de 1×1, aportando amplitud al periodo y aleatoriedad a la secuencia, con un coste computacional muy bajo y comportándose con absoluta normalidad estadística.

3.3 Mejora del sistema de extracción

Con el objetivo de aumentar el rendimiento, resulta interesante establecer un nuevo sistema de extracción que maximice en lo posible la cantidad de *bpi* y que aproveche toda la aleatoriedad que los elementos de los bloques X, Y y Z puedan proporcionar.

Este método de extracción se basa en extraer de cada elemento de estos bloques más de un *bms (*dos, tres, cuatro...). Para aprovechar varios *bms* sin condicionar estadísticamente los resultados de la secuencia extraída, se debe escoger un primo p de forma que todos los bits a utilizar de los elementos de $\mathbb{Z}_p$ tengan aproximadamente la misma probabilidad de ser 0 o 1 en su representación binaria.

Cualquier bit de la codificación binaria de los elementos de un conjunto es equiprobable si el cardinal del conjunto es 2^n y se usan n bits para su codificación. Para el caso de los elementos de $\mathbb{Z}_p$, si seguimos el mismo razonamiento, los bits resultaran más equiprobables en su codificación binaria cuanto más se aproxime p a una potencia de 2.

En concreto, los primos con la forma $p = 2^n + 1$, con $n \in \mathbb{Z}$ suficientemente grande, proporcionan elementos de $\mathbb{Z}_p$ con probabilidades próximas a ½ para 0 y 1 en sus n *bms*. Esto ocurre porque todos los elementos tienen una combinación distinta de los n *bms* en su codificación binaria, salvo en el caso de los valores 0 y 2^n que tienen iguales los n *bms* (todos a 0).

Obsérvese que son $n + 1$ bits los necesarios para codificar los elementos de de $\mathbb{Z}_p$ con $p = 2^n + 1$, y que si se usan n *bms*, queda solo el bit más significativo por utilizar, que es 0 para todos los elementos de $\mathbb{Z}_p$ salvo para $p - 1$ que es 1.

Por este motivo, se han probado sistemas de extracción con $n = 1$, 2, 4, 8 y 16 *bms*, ya que para estos valores de n se tiene que $p = 2^n + 1$ es primo.

bms	p	$P(\{bit = 0\})$	$P(\{bit = 1\})$
1	2	0,5	0,5
1	3	0,66666667	0,33333333
2	5	0,6	0,4
4	17	0,52941176	0,47058824
8	257	0,50194553	0,49805447
16	65537	0,50000763	0,49999237

Tabla 3.3: Probabilidades al bit

Para $n = 1$, el primo 3 se ha sustituido por el primo 2 que proporciona un bit exactamente equiprobable, como se muestra en la tabla 3.3 donde aparece la probabilidad de que cualquiera de los n *bms* sean 0 o 1, para los primos elegidos.

Así pues, los primos para esta segunda batería de pruebas han sido 2, 5, 17, 257 y 65537.

Universitat d'Alacant
Universidad de Alicante

Definición 3.16: Bits empaquetados

Una secuencia de bits empaquetada es aquella secuencia que se proporciona o almacena en bytes, formando todos los bits de cada byte parte de la secuencia.

Definición 3.17: Bits desempaquetados

Una secuencia de bits está desempaquetada cuando se proporciona o almacena cada bit en un byte, usándose sólo el *bms* de cada byte, y poniendo a cero el resto.

En la anterior batería de pruebas, las secuencias se han generado desempaquetadas. En los nuevos experimentos resulta más rentable, en los casos de tener que extraer 8 y 16 *bms* de cada elemento, usar secuencias empaquetadas, extrayendo directamente uno o dos bytes por elemento.

Así pues, se usaran dos nuevos tipos de extracción que se definen a continuación:

Definición 3.18: Extracción *byte*

Se entiende por sistema de extracción *byte* a aquel que extrae, de cada uno de los elementos de los bloques superiores $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, los 8 bms directamente como un solo byte.

Definición 3.19: Extracción *double*

El sistema de extracción *double* aquel que, de cada uno de los elementos de los bloques superiores $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, extrae los 16 bms directamente como dos bytes.

De esta forma, dependiendo del primo y tipo de extracción usado en la parametrización, las secuencias de esta nueva batería de experimentos se han generado empaquetadas o desempaquetadas. Los test utilizados están preparados para trabajar igualmente de una forma u otra.

Los primos 2 y 5 generan secuencias desempaquetadas, mientras que los primos 17, 257 y 65537 generan secuencias empaquetadas. Para 2 se ha usado la antigua extracción *fast* de un bit por elemento, y para 5 y 17 se han usado operaciones adicionales posteriores la extracción *byte* para obtener la secuencia desempaquetada o empaquetada. Para 257 se ha usado la extracción *byte* y para 65537 la *double*.

Los tamaños de bloques utilizados hasta ahora se pueden considerar relativamente pequeños. Como se ha indicado con anterioridad, uno de los inconvenientes a la hora de emplear bloques más grandes es la dificultad para hallar polinomios primitivos de grados grandes, sobre todo si el primo también es grande.

Antes de diseñar la nueva batería de experimentos se ha tenido que ampliar la tabla de polinomios y raíces primitivas, siendo necesario realizar una nueva búsqueda de polinomios primitivos.

Esta vez se ha usado la librería de cálculo de Sean Erik O'Connor [75] y su utilidad Primpoly Version 12.0 ("A Program for Computing Primitive Polynomials"). Algunos de los polinomios han sido localizados tras largos periodos de computación.

3.3.1 Diseño de la segunda batería de experimentos

Para los primos finalmente escogidos, la disponibilidad de polinomios en la nueva tabla ampliada, permite establecer tres grupos de tamaños que se han

denominado *small*, *medium* y *large* (S, M y L). Los tamaños 2, 3 y 5 se consideran tamaños del grupo S (próximos a 2^2), mientras que 13, 15 y 19 son M (próximos a 2^4) y 59, 64 y 67 son L (próximos a 2^6).

p	2			5			17			257			65537		
	r	s	t	r	s	t	r	s	t	r	s	t	r	s	t
1SS													1	2	7
SSS													2	3	5
1SM										1	2	19	1	2	19
1SM+										1	5	19	1	5	19
SSM										2	3	15	2	3	15
SSM+										3	5	19	3	5	19
1MM							1	15	19	1	15	19	1	15	19
SMM							5	15	19	5	15	19	5	15	19
MMM							13	15	19	13	15	19	13	15	19
1ML				1	19	59	1	19	58	1	19	59			
SML				5	19	59	5	19	58	5	19	59			
MML				15	19	59	15	19	58	15	19	59			
1SL				1	5	59	1	5	58	1	5	59			
SSL				2	5	59	2	5	58	2	5	59			
1SL+				1	2	67	1	2	67						
SSL+				2	3	67	2	3	67						
1LL	1	61	67	1	61	67	1	61	67						
SLL	5	61	67	5	61	67	5	61	67						
MLL	15	61	67	15	61	67	15	61	67						
LLL	59	61	67	59	61	67	58	61	67						

Tabla 3.4: Combinaciones probadas

Universitat d'Alacant
Universidad de Alicante

Así, las combinaciones a probar las denominaremos por la combinación de los grupos a los que pertenezcan los tamaños. El bloque de tamaño 1 se especifica explícitamente en la denominación, como un grupo propio. Algunas combinaciones de grupos se repiten, en tal caso, para distinguirlas se le añade un + a la denominación de la combinación que tiene algún tamaño de bloque mayor. En tabla 3.4 se indica qué combinaciones se han escogido para esta batería de experimentos.

Obsérvese que, para el primo 17, en las combinaciones en las que era necesario emplear el tamaño de bloque 59 se ha utilizado un bloque de tamaño 58 debido a que no se dispone de un polinomio primitivo de grado 59 para dicho primo.

En total se ha estudiado 4 combinaciones para el primo 2, 11 para el 5, 14 para el 17, 12 para el 257 y 9 para el 65537; sumando 50 combinaciones, todas ellas admisibles (véase la restricción (3.66)). Cabe destacar que, para hacer

admisible la combinación 1SS con el primo 65537, ha sido necesario que el tercer bloque (C) fuese de tamaño 7

En la tabla 3.5 se muestra, para cada combinación y primo, el mínimo de bits necesarios para expresar el periodo y la semilla, siempre mayor o igual a 128 bits.

	mbp					*mbs*				
	2	5	17	257	65537	2	5	17	257	65537
1SS					128					368
SSS					128					496
1SM				160	320				472	944
1SM+				184	368				952	1904
SSM				128	256				648	1296
SSM+				200	400				1336	2672
1MM			134	264	528			1303	2553	5104
SMM			134	264	528			1859	3642	7280
MMM			184	360	720			2971	5820	11632
1ML		179	310	616			2783	4819	9598	
SML		188	327	648			3508	6078	12096	
MML		211	368	728			5319	9225	18340	
1SL		146	253	504			833	1442	2874	
SSL		149	253	512			982	1700	3386	
1SL+		158	278				471	829		
SSL+		163	286				791	1393		
1LL	128	295	519			4215	9786	17228		
SLL	132	304	535			4727	10975	19321		
MLL	142	328	576			6007	13947	24553		
LLL	186	430	752			11639	27024	47050		

Universitat d'Alacant
Universidad de Alicante

Tabla 3.5: Mínimo de bits para el periodo y la semilla por combinación

El sistema de extracción elegido para cada primo consigue aprovechar al máximo la aleatoriedad proporcionada por los bloques superiores $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$.

Se han extraído secuencias de 200.000 bits de forma que el generador itere un mínimo de 5 veces ($h = 6$), ya que en algunas combinaciones el sistema de extracción obtiene más de 45.000 bits por iteración.

En la tabla 3.6 se indican los *bpi* de cada combinación y cuantas potencias de M (iteraciones) son necesarias para obtener los 200.000 bits.

	bpi					*Iteraciones para 200000 bits*				
	2	5	17	257	65537	2	5	17	257	65537
1SS					368					544
SSS					496					404
1SM				472	944				424	212
1SM+				952	1904				211	106
SSM				648	1296				309	155
SSM+				1336	2672				150	75
1MM			1276	2552	5104			157	79	40
SMM			1820	3640	7280			110	55	28
MMM			2908	5816	11632			69	35	18
1ML		2398	4716	9592			84	43	21	
SML		3022	5948	12088			67	34	17	
MML		4582	9028	18328			44	23	11	
1SL		718	1412	2872			279	142	70	
SSL		846	1664	3384			237	121	60	
1SL+		406	812				493	247		
SSL+		682	1364				294	147		
1LL	4215	8430	16860			48	24	12		
SLL	4727	9454	18908			43	22	11		
MLL	6007	12014	24028			34	17	9		
LLL	11639	23278	46044			18	9	5		

Tabla 3.6: Bits por iteración y número de iteraciones por combinación

La combinación con bloques más pequeños (1SS) para el primo 65537 es la que más iteraciones necesita realizar para conseguir los 200.000 bits, mientras que la de tamaños de bloques más grandes, con el primo 17 es la que menos iteraciones realiza. Evidentemente, el coste por iteración no será igual con bloques de tamaño 1, 2 y 7 que con bloques de 58, 61, y 67.

Previamente al lanzamiento de la batería de experimentos, y obtención de resultados, se ha realizado una estimación del coste por iteración para cada combinación, considerando por igual operaciones de suma, multiplicación y resto (módulo p).

Esa estimación del coste de iteración en función de r, s y t es

$$C(r,s,t) = (r^2s + rs^2 + rs) + (r^2t + rst + rt^2 + rt\,) + (s^2t + st^2 + st)\; + (r^3 + r^2)\; + (s^3 + s^2)\,, \tag{3.78}$$

que se corresponde con los cálculos necesarios para $Y^{(h)}$, $X^{(h)}$, $Z^{(h)}$, A^h y B^h.

En la tabla 3.7 se indica la estimación del coste por iteración de cada combinación, y lo que es más interesante, la estimación del coste por bit.

	Estimación de coste por iteración					*Estimación de coste por bit*				
	2	5	17	257	65537	2	5	17	257	65537
1SS					239					0,65
SSS					329					0,66
1SM				1295	1295				2,74	1,37
1SM+				3056	3056				3,21	1,61
SSM				1569	1569				2,42	1,21
SSM+				4292	4292				3,21	1,61
1MM			14516	14516	14516			11,38	5,69	2,84
SMM			19100	19100	19100			10,49	5,25	2,62
MMM			33452	33452	33452			11,50	5,75	2,88
1ML		100900	98159	100900			42,08	20,81	10,52	
SML		123084	119771	123084			40,73	20,14	10,18	
MML		192544	187661	192544			42,02	20,79	10,51	
1SL		23256	22517	23256			32,39	15,95	8,10	
SSL		27323	26458	27323			32,30	15,90	8,07	
1SL+		14159	14159				34,87	17,44		
SSL+		24137	24137				35,39	17,70		
1LL	770480	770480	770480			182,79	91,40	45,70		
SLL	823400	823400	823400			174,19	87,10	43,55		
MLL	976700	976700	976700			162,59	81,30	40,65		
LLL	2145428	2145428	2107643			184,33	92,17	45,77		

Tabla 3.7: Estimación de los costes por iteración y por bit

Como se puede apreciar, los resultados confirman la hipótesis de que el algoritmo resulta más eficiente conforme sea mayor el primo, pero menor el tamaño de los bloques.

Universidad de Alicante

Al igual que en la batería anterior, todos los experimentos (50) han sido repetidos 10 veces con semillas distintas, identificadas de 1 al 10, para poder repetir las mismas secuencias con y sin filtrado XOR. En total se han generado y analizado 1.000 secuencias de 200.000 bits cada una.

Se han medido los ciclos de procesador en cada iteración y, en ambos casos, el coste por bit en ciclos reales de procesador se reducía con el aumento del primo y la reducción del tamaño de los bloques, con una tendencia mayor todavía a la esperada.

Inicialmente, sólo se tenía en cuenta el cálculo de los bloques y no el proceso de extracción y filtrado. En la práctica la medición se ha realizado por separado, distinguiendo así los ciclos de procesador imputables a uno u otro proceso en cada iteración. La extracción y filtrado ha supuesto un coste siempre mayor, aumentando más todavía que la del cálculo de los bloques al aumentar el tamaño de estos; justificando, así, que la tendencia a aumentar el coste con el tamaño de los bloques haya sido todavía mayor que la esperada.

Este nuevo sistema de extracción obtiene un número de bits por operación proporcional tanto al tamaño del primo como de los bloques. No obstante, aumentar el tamaño del primo no supone un aumento de coste adicional, mientras que aumentar el tamaño de los bloques sí que produce un incremento del coste computacional.

Estos resultados, al igual que los referentes a los de los test estadísticos, pueden consultase en el apéndice A.2.

En los test estadísticos se puede apreciar que los primos bajos muestran un sesgo destacable, en todos los test salvo en el de autocorrelación y complejidad lineal. Con el filtrado los resultados mejoran pero no resultan totalmente satisfactorios.

El primo 65537 es el menos afectado, seguido por el 257 y el 2. Entre las combinaciones escogidas para estos tres primos, las combinaciones con bloques de menor tamaño son las más afectadas.

El hecho de que los primos 5 y 17 sean los más afectados por el sesgo, precisamente los que menos equiprobabilidad proporcionan a los bits extraídos (véase la tabla 3.3), hace pensar que esta pueda ser la causa de las anomalías estadísticas que el filtrado *XOR* no parece ser capaz de erradicar.

3.3.2 Variante en los sistemas de extracción.

Con el objetivo de solventar los problemas estadísticos encontrados, se ha diseñado sistemas de extracción *byte* y *double* modificados que fuerzan la equiprobabilidad de los bits.

Universidad de Alicante

Definición 3.20: Extracción con *zero replacement* (*0r*)

Un sistema de extracción *byte* o *doble* con *zero replacement* es aquel sistema de extracción *byte* o *double* que sustituye las extracciones pares (o impares) de un byte o doble byte con todos sus bits a ceros por la de la codificación en binario de los n *bms* del siguiente elemento de una secuencia cíclica de todos los elementos de $\mathbb{Z}_p - \{0\}$. En consecuencia, la mitad de los elementos 0 y $p-1$ de los bloques superiores $X^{(h)}$, $Y^{(h)}$ y $Z^{(h)}$, son interpretados por, y exclusivamente, el proceso de extracción, como elementos entre 1 y $p-1$.

Estadísticamente, la codificación con n *bms* todos a cero tiene el doble de probabilidades de aparecer que cualquier otra; al coincidir esos n *bms* tanto en la codificación del valor 0 como en la del valor $p-1$. La modificación *0r* al sistema de extracción hace que finalmente la probabilidad de que cualquier bit sea 0 o 1 resulte equiprobable.

La secuencia cíclica de todos los elementos de $\mathbb{Z}_p$ se obtiene generándolos a partir de un elemento escogido al azar como semilla y multiplicándolo sucesivamente por una raíz primitiva. El elemento semilla se vincula al identificador de semilla usado para el generador, y la raíz primitiva se obtiene de la tabla polinomios primitivos.

La batería de experimentos se ha repetido con esta modificación del sistema de extracción, salvo para el primo 2, para el que no tiene sentido la nueva estrategia. Resultan 46 combinaciones con y sin filtrado XOR, repetida cada una para las 10 semillas asociadas a los 10 primeros identificadores; analizándose, en total, 920 secuencias de 200.000 bits que se han comparado entre ellas y sus equivalentes anteriores.

Como se puede observar en los resultados del apéndice A.3, la modificación *0r* únicamente mejora los test deficientes de forma significativa en los casos en los que además se aplica filtrado XOR, y aun así no resulta aceptable para el primo 5 y algunos test del primo 17. La falta de equiprobabilidad en los *n bms* extraídos no parece ser la causa principal del sesgo en los test de frecuencia, serial, póker y rachas.

Además, este sistema introduce tiempos de ejecución variable en función del número de elementos encontrados con todos los *bms* a cero y, por lo tanto, del estado inicial del generador; pudiendo, eventualmente, resultar en una vulnerabilidad que facilitase la recuperación de la semilla mediante el análisis de los tiempos de generación de la secuencia.

Universitat d'Alacant
Universidad de Alicante

3.4 Mejoras en el filtrado

Dados los resultados observados, se ha decidido trabajar exclusivamente con el primo 65537 en las siguientes baterías de experimentos. Ha quedado probado que usar un primo alto resulta ventajoso tanto para el rendimiento del algoritmo como para obtener los mejores resultados estadísticos.

Respecto a usar bloques grandes o pequeños, la rentabilidad y los buenos resultados estadísticos no marcan la misma dirección de búsqueda. Mientras que los bloques pequeños generan secuencias con menos coste computacional, son los bloque grandes los que proporcionan mejores resultados en los test estadísticos. Por otra parte, uno de los inconvenientes de usar un primo alto es la dificultad para encontrar polinomios primitivos; lo que resulta un problema todavía mayor en el caso de bloques grandes, que requieren polinomios primitivos de grados mayores.

Por estos motivos, se ha diseñado un nuevo un sistema de filtrado que permite usar bloques pequeños junto con primos grandes sin comprometer los resultados estadísticos de las secuencias generadas. Este sistema de filtrado aporta, además, una serie de ventajas sobre los usados anteriormente, las cuales se comentan a lo largo de esta sección.

3.4.1 Cajas de sustitución

El nuevo sistema de filtrado es similar al descrito en [6-8], que se basa en cajas de sustitución (*s-box*) 32×32 inspiradas en el esquema de inicialización y actualización de la *s-box* interna que utiliza RC4 (véanse [46, 81]).

Definición 3.21: Caja de sustitución (*s-box*)

Una caja de sustitución es una tabla en la que se asigna un valor de salida a cada valor de entrada posible. En general, una *s-box* de tamaño $m \times n$ tiene m bits de entrada y n bits de salida; se suele implementar como una tabla de 2^m entradas de n bits cada una.

En este caso, la *s-box* 32×32 en realidad se aproxima mediante una combinación de cuatro cajas 8×32, en las que a cada byte de entrada se le asocian 32 bits de salida. La sustitución de cuatro bytes (32 bits) es el resultado de una operación XOR con las salidas correspondientes de cada una de las cuatro cajas 8×32 para cada uno de los cuatro bytes respectivamente, proporcionando así una sustitución de 32×32 bits. Esta aproximación resulta necesaria dado el espacio necesario para almacenar una caja de 32×32 real (2^{34} bytes frente a los 2^{10} bytes requeridos por las 4 cajas de 8×32).

Definición 3.22: Filtrado con 32×32 *s-box*

Consiste en realizar una operación XOR entre los últimos 32 bits de la secuencia filtrada y los 32 bits salientes de una caja de sustitución 32×32, obtenidos a partir de los siguientes cuatros bytes (32 bits) de secuencia a filtrar.

El filtrado con *s-box* 32×32 opera de forma similar a como actúa el filtro XOR salvo que, en vez de actuar bit a bit, o byte a byte, opera de 4 en 4 bytes tras su substitución con una *s-box* de 32×32.

La inicialización de la *s-box* se realiza a partir de la clave usada para el cifrado en flujo. A partir de esta clave se obtiene la *s-box*, y mediante la clave y la propia *s-box* se inicializa la semilla del generador (valores iniciales de los bloques X, Y, Z).

A partir de esta nueva batería de experimentos, en vez de usar identificadores de semilla, se ha usado identificadores de clave. Las claves, de

256 bytes, se han generado mediante un identificador, de forma similar a como sucedía en anteriores experimentos con los identificadores de semilla. El identificador asociado a la clave permite repetir el experimento y obtener de nuevo la misma secuencia a analizar.

Si bien este esquema permite el uso de claves de cualquier tamaño, las claves consideradas (de 256 bytes o 2048 bits) exceden con creces los niveles de seguridad recomendados en la actualidad para cifradores en flujo (256 bits).

Ha sido necesario desarrollar una nueva versión del código fuente correspondiente al algoritmo (véase el apéndice B.2) debido a las extensas modificaciones realizadas como la incorporación de la *s-box* en el filtrado, el uso de identificadores de clave en lugar de identificadores de semilla o que la búsqueda de parámetros se haya reducido exclusivamente a combinaciones con el primo 65537. El código correspondiente a la versión original se encuentra en el apéndice B.1.

Para la construcción de la *s-box* de 32×32 se ha empleado una *s-box* temporal de 8×8 que contiene, inicialmente, los valores del 0 al 255 ordenados y se van intercambiando en función de los bytes de la clave (que se van repitiendo de forma cíclica). Así, esta *s-box* temporal va evolucionando y cada 256 intercambios se genera una nueva caja de 8×8, concatenando 4 de ellas para formar una *s-box* de 8×32; son necesarias, por lo tanto, 16 *s-box* de 8×8 para obtener las 4 *s-box* de 8×32 que conforman el filtro dependiente de la clave. En las líneas de la 402 a la 439 del código fuente *motor.c*, disponible en el apéndice B.2, se puede consultar, en lenguaje C, el algoritmo para la construcción de la *s-box* 32×32 usada.

La clave no determina únicamente la *s-box*, sino también la semilla. El algoritmo que determina la semilla en función de la clave consiste en obtener valores para cada elemento de los bloques X, Y y Z a partir de los 16 primeros bytes de la clave, la *s-box* 32×32 generada previamente y la posición de cada elemento.

Los elementos de la semilla se rellenan por orden de fila y columna: primero los del bloque Y, luego los del bloque X y por último los del bloque Z. Para evitar que los bloques Y y Z sean nulos, el primer y último elemento de la semilla ($y_{1,1}$ y $z_{s,t}$) son inicializados a dos constantes arbitrarias cuyos valores en hexadecimal son $0x5A5A$ y $0xA5A5$. En las líneas de la 454 a la 475 del código fuente *motor.c* (véase en el apéndice B.2) se muestra en lenguaje C cómo se realiza exactamente la generación de la semilla.

3.4.2 Ventajas del filtro *s-box* 32×32

La construcción de la *s-box* y relleno de la semilla añaden al nuevo algoritmo un proceso de inicialización previo a la iteración de los procesos de cálculo de las diferentes potencias de la matriz M, extracción y filtrado de bits.

A este proceso inicial se le ha añadido una serie de iteraciones iniciales en blanco; en concreto, 64 iteraciones del cálculo de potencias de M en las que no se produce ningún bit de salida. Esto dificulta el criptoanálisis, al omitir la parte inicial de la secuencia (que tiene una relación más directa con la semilla) y, también, permite que los bloques de la diagonal sean matrices menos dispersas (véase la expresión (2.60)) al iniciar la generación de la secuencia de salida real.

Por otra parte, con esta incorporación, el algoritmo es más versátil desde el punto de vista criptográfico al incorporar el concepto de clave, a partir de la cual se obtiene la semilla adecuada, sin bloques Y o Z nulos; permitiendo, también, dificultar el criptoanálisis mediante un ataque por fuerza bruta debido al coste asociado a la inicialización del algoritmo.

El filtro con *s-box* 32×32 es un filtro no lineal [6-8, 43, 49, 71], necesario para evitar tanto el criptoanálisis diferencial como el lineal dado que el resto del generador se fundamenta en operaciones lineales. Este tipo de criptoanálisis busca aproximar el generador mediante un sistema de ecuaciones lineales obteniendo, en algunos casos, reducciones significativas en el tiempo de búsqueda por fuerza bruta. Además, por la forma en la que se construyen las cajas 8×8 que lo forman, es un filtro balanceado, por lo que no perjudica las características estadísticas originales de la secuencia a filtrar.

Universitat d'Alacant
Universidad de Alicante

3.4.3 Batería de experimentación

En esta versión del generador ha usado las mismas combinaciones que en la batería anterior para el primo 65537. En la tabla 3.8 se muestra las combinaciones admisibles (*mbp* y *mbs* de al menos 128) que han sido probadas.

Son 9 combinaciones con un análisis mucho más exhaustivo que en las ocasiones anteriores. Se han analizado 100 secuencias de 200.000 bits por cada combinación, sumando 900 pruebas en total.

	p = 65537			16 bits por elemento			Iteraciones para 200000 bits
	r	*s*	*t*	*bpi*	*mbp*	*mbs*	
1SS	1	2	7	368	128	368	544 (+64)
SSS	2	3	5	496	128	496	404 (+64)
1SM	1	2	19	944	320	944	212 (+64)
1SM+	1	5	19	1904	368	1904	106 (+64)
SSM	2	3	15	1296	256	1296	155 (+64)
SSM+	3	5	19	2672	400	2672	75 (+64)
1MM	1	15	19	5104	528	5104	40 (+64)
SMM	5	15	19	7280	528	7280	28 (+64)
MMM	13	15	19	11632	720	11632	18 (+64)

Tabla 3.8: Combinaciones admisibles probadas para 65537

Los resultados, expresados como el porcentaje de secuencias rechazadas, se encuentran el apéndice A.3. Se puede observar que son muy satisfactorios para el nivel de significación menos restrictivo (un 0,1% de probabilidad de rechazar una secuencia que sí sea aleatoria), y admisibles para el nivel de significación más restrictivo (un 10% de probabilidad). Respecto a los tiempos obtenidos, se ha verificado que son las combinaciones con bloques más pequeños las que obtienen un menor coste computacional por bit.

3.5 Optimización el algoritmo

Una vez alcanzado un nivel de aleatoriedad aceptable en los test estadísticos sencillos, surge la necesidad de probar secuencias mucho más largas con test más exigentes.

Con el objetivo de experimentar con secuencias mucho más largas, se ha decidido acotar los experimentos a las combinaciones que proporcionan un mayor rendimiento: las correspondientes con los tamaños 1×1 y 2×2 en los bloques A y B, respectivamente. En este caso, el tamaño 7×7 del bloque C es el más pequeño que resulta en una combinación admisible, mientras que el tamaño 19×19 es el más grande para el que se dispone de polinomio primitivo.

De entre esos dos valores de t, se ha probado todas las combinaciones de las que se disponen polinomios primitivos ($t = 7, 8, \ldots, 15$ y $t = 19$ con $r = 1$ y $s = 2$) con la suite de test PractRand (véase la sección 1.3.4.1) con filtro *s-box*, y cuatro combinaciones ($t = 7, 13, 15$ y 19) con la versión del algoritmo sin filtro. Se pueden ver en los resultados incluidos en el apéndice A.4

Las secuencias generadas sin filtro son rechazadas por PractRand. El fallo se produce siempre antes de alcanzar tamaños de secuencia de 64 o 128 megabytes, dependiendo de cuál de las combinaciones se trate y de la semilla escogida. Por si el problema se debiera a un sesgo en las primeras iteraciones del algoritmo, se han ampliado las iteraciones en blanco hasta 4.000.000 de iteraciones; lo que equivale, en el caso de la combinación de $r = 1$, $s = 2$ y $t = 7$, a omitir más de 175 megabytes de la secuencia y casi tres veces más en la combinación con $t = 19$. A pesar de ello, las secuencias sin filtro no han superado una longitud de 128 megabytes antes de ser rechazadas por PractRand.

Sin embargo, con el filtrado con *s-box* 32×32, todas combinaciones y secuencias probadas han sido aceptadas, llegando a alcanzar secuencias de hasta 1 terabyte (2^{43} bits) de longitud sin rechazo alguno; como se puede observar en los resultados que se muestran en el apéndice A.4. Con estas pruebas queda plenamente justificada la conveniencia de usar el filtro con *s-box* 32×32.

Respecto al coste computacional del algoritmo, se ha comparado la velocidad de generación de la secuencia cifrante para las combinaciones 7 (r=1, s=2 y t=7) y 19 (r=1, s=2 y t=19) con la de otros cifradores en flujo populares en la actualidad: AES (en modo OFB), RC4, Salsa20 y HC128 (véase sección 1.5). La implementación de dichos algoritmos se ha realizado sin hacer uso de optimizaciones específicas, ensamblador o aceleración por hardware, para hacer la comparativa lo más justa posible; se ha ejecutado sobre un procesador Intel i7 920 de 2,67 GHz con Windows 8.1; y se han generado las secuencias en memoria, evitando volcar datos al sistema de ficheros.

Como se puede apreciar en la tabla 3.9, la velocidad de las combinaciones 1-2-7 y 1-2-19 es similar a la de AES (OFB), siendo la velocidad de la combinación 1-2-7 algo mejor que la de la combinación 1-2-19. Aunque la diferencia es mínima entre pruebas de la misma combinación, las velocidades que se indican en la tabla son las mejores obtenidas al generar las secuencias de un gigabyte con diferentes claves para cada combinación. Es esperable que las velocidades de otras combinaciones 1-2-n, con n entre 7 y 19, sean similares.

Lamentablemente, en ningún caso las combinaciones estudiadas resultan competitivas respecto a los algoritmos RC4, Salsa20 y HC128; que son entre dos y tres veces más rápidos.

AES	RC4	Salsa20	HC128	1-2-7	1-2-19
60 MB/s	154 MB/s	138 MB/s	131 MB/s	68 MB/s	53 MB/s

Tabla 3.9: Comparativa velocidades combinaciones 1-2-7 y 1-2-19

Para obtener mayor velocidad, resulta necesario el análisis de los cálculos a efectuar, optimizando las sumas con ceros y productos por ceros o unos que puedan surgir.

3.5.1 Enfoque optimizado

Es importante considerar que el único elemento en el bloque A es, en realidad un generador de $\mathbb{Z}_p$, y que los bloques B y C se crean a partir de polinomios primitivos en $\mathbb{Z}_p[x]$ que, en este caso (véase el apéndice C), son trinomios mónicos definidos por los coeficientes del término independiente y el término en x.

Si $p-a_0$ es una raíz primitiva de $\mathbb{Z}_p$, y $x^2 + b_1x + b_0$ y $x^d + c_1x + c_0$ son polinomios primitivos de grado 2 y d, con coeficientes en $\mathbb{Z}_p$, entonces, las seis constantes

$$\begin{aligned} k_0 &= p\,, \\ k_1 &= p - a_0\,, \\ k_2 &= p - b_0\,, \\ k_3 &= p - b_1\,, \\ k_4 &= p - c_0\,, \\ k_5 &= p - c_1\,, \end{aligned} \tag{3.79}$$

constituyen una 6-tupla $K = (k_0, \ldots, k_5)$ que, junto con el grado d, determinan el subconjunto $\Theta_3^{(d,K)} \subset \Theta_3$ cuyas matrices $(d+3) \times (d+3)$ tienen la forma

$$M = \begin{bmatrix} [k_1] & [s_0 \;\; s_1] & \begin{bmatrix} s_2 & s_3 & s_4 & \cdots & s_d & s_{d+1} \end{bmatrix} \\ 0 & \begin{bmatrix} 0 & 1 \\ k_2 & k_3 \end{bmatrix} & \begin{bmatrix} s_{d+2} & s_{d+3} & s_{d+4} & \cdots & s_{2d} & s_{2d+1} \\ s_{2d+2} & s_{2d+3} & s_{2d+4} & \cdots & s_{3d} & s_{3d+1} \end{bmatrix} \\ 0 & 0 & \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ k_4 & k_5 & 0 & \cdots & 0 & 0 \end{bmatrix} \end{bmatrix}, \tag{3.80}$$

y sus elementos pertenecen a $\mathbb{Z}_{k_0}$.

Las matrices iniciales M construidas para las combinaciones 1-2-d admisibles del primo $p = 65537$ tienen la forma de la expresión (3.80) y pertenecen a $\Theta_3^{(d,K)}$, donde las constantes de la 6-tupla K son $k_0 = 65537$, $k_1 = 3$ (la raíz primitiva usada para $\mathbb{Z}_{k_0}$), $k_2 = 65534$ y $k_3 = 65536$ (calculadas a partir del polinomio primitivo de segundo grado $x^2 + x + 3$), y k_4 y k_5 que vienen determinadas por los polinomios primitivos de grado d que se indica en la tabla 3.10.

d	Pol. Prim.	k_4	k_5
7	$x^7 + x + 3$	65534	65536
8	$x^8 + x + 20$	65517	65536
9	$x^9 + x + 10$	65527	65536
10	$x^{10} + x + 6$	65531	65536
11	$x^{11} + x + 27$	65510	65536
12	$x^{12} + x + 11$	65526	65536
13	$x^{13} + x + 56$	65481	65536
14	$x^{14} + x + 108$	65429	65536
15	$x^{15} + x + 20$	65517	65536
19	$x^{19} + x + 47$	65490	65536

Tabla 3.10: Valores de k_4 y k_5 para distintos valores de d

En realidad el valor de k_5 que se usa es constante $(p-1)$ dado que todos los polinomios primitivos usados tienen coeficiente 1 en el término en x. Para las combinaciones de tamaño $d = 16$, 17 y 18 no se dispone de polinomio primitivo, por lo que no se incluyen entre las probadas.

La (3d+2)-tupla $S = (s_0, \dots, s_{3d+1})$, formada por los valores de los elementos de los bloques superiores en la expresión (3.80), constituye la semilla. De forma similar, se considera la (3d+2)-tupla $P^{(h)} = (p_0^{(h)}, \dots, p_{3d+1}^{(h)})$ formada por los valores de los elementos de los bloques superiores de

$$M^h = \begin{bmatrix} \begin{bmatrix} v_0^{(h)} \end{bmatrix} & \begin{bmatrix} p_0^{(h)} & p_1^{(h)} \end{bmatrix} & \begin{bmatrix} p_2^{(h)} & p_3^{(h)} & p_4^{(h)} & \cdots & p_d^{(h)} & p_{d+1}^{(h)} \end{bmatrix} \\ 0 & \begin{bmatrix} v_1^{(h)} & v_2^{(h)} \\ v_3^{(h)} & v_4^{(h)} \end{bmatrix} & \begin{bmatrix} p_{d+2}^{(h)} & p_{d+3}^{(h)} & p_{d+4}^{(h)} & \cdots & p_{2d}^{(h)} & p_{2d+1}^{(h)} \\ p_{2d+2}^{(h)} & p_{2d+3}^{(h)} & p_{2d+4}^{(h)} & \cdots & p_{3d}^{(h)} & p_{3d+1}^{(h)} \end{bmatrix} \\ 0 & 0 & \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ k_4 & k_5 & 0 & \cdots & 0 & 0 \end{bmatrix}^h \end{bmatrix}, \quad \textbf{(3.81)}$$

siendo $v_0^{(h)} = k_1^h$ y $\begin{bmatrix} v_1^{(h)} & v_2^{(h)} \\ v_3^{(h)} & v_4^{(h)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ k_2 & k_3 \end{bmatrix}^h$.

A partir de la función $f_{(d,K)}: \mathbb{Z}_{k_0}^{3d+2} \to \mathbb{Z}_{k_0}^{3d+2}$ tal que $f_{(d,K)}(P^{(i)}) = P^{(i+1)}$, se define la función iterada $f_{(d,K)}^h: \mathbb{Z}_{k_0}^{3d+2} \to \mathbb{Z}_{k_0}^{3d+2}$ como la función $f_{(d,K)}$ compuesta consigo misma h veces. Como $P^{(1)} = S$, entonces

$$
\begin{aligned}
&f_{(d,K)}(S) = f_{(d,K)}\left(P^{(1)}\right) = P^{(2)}\,, \\
&f_{(d,K)}^{2}(S) = f_{(d,K)} \circ f_{(d,K)}(S) = f_{(d,k)}\left(P^{(2)}\right) = P^{(3)}\,, \\
&f_{(d,K)}^{3}(S) = f_{(d,K)} \circ f_{(d,K)}^{2}(S) = f_{(d,k)}\left(P^{(3)}\right) = P^{(4)}\,, \\
&\vdots \\
&f_{(d,K)}^{h}(S) = f_{(d,K)} \circ f_{(d,K)}^{h-1}(S) = f_{(d,k)}\left(P^{(h)}\right) = P^{(h+1)}\,.
\end{aligned}
\tag{3.82}
$$

Debido a que $\Theta_3^{(d,K)}$ es un subconjunto de Θ_3, todas las propiedades descritas anteriormente, son heredadas por el nuevo planteamiento, de manera que se puede afirmar que la función iterada $f_{(d,K)}^{h}$ tiene una órbita periódica cuyo periodo coincide con el grado de $M \in \Theta_3^{(d,K)}$.

La secuencia de bits se obtiene al aplicar el sistema de extracción (y posterior filtrado), para cada iteración h, a la tupla $P^{(h)}$ obtenida por $f_{(d,K)}^{h-1}(S)$, para $h \geq 2$ (o en su defecto $h \geq 65$ si el generador se inicializa con 64 iteraciones sin producción).

De las expresiones (2.28), (2.44), (2.57), (2.58), (2.59), (3.80) y (3.81), se deduce que la (3d+2)-tupla $P^{(h+1)}$ se puede expresar como una función de la semilla inicial, S; la (3d+2)-tupla $P^{(h)}$ y los valores $v_0^{(h)}, v_1^{(h)}, v_2^{(h)}, v_3^{(h)}, v_4^{(h)}$. Además, que no es necesario conocer la potencia del último bloque $d \times d$ de la diagonal de M^h. Es decir, los valores de la 5-tupla $V^{(h)} = (v_0^{(h)}, v_1^{(h)}, v_2^{(h)}, v_3^{(h)}, v_4^{(h)})$, junto con la última (3d+2)-tupla calculada, $P^{(h)}$, forman el estado interno del generador, y de ese estado interno, y la semilla inicial, S, depende el resultado de la siguiente iteración.

Efectuando el producto matricial de M^h, en la forma (3.82), por M, en su forma (3.81), se obtienen las siguientes expresiones para los valores de las componentes de la (3d+2)-tupla $P^{(h+1)}$.

Universitat d'Alacant
Universidad de Alicante

$$
\begin{aligned}
p_0^{(h+1)} &= (v_0^{(h)} s_0 + p_1^{(h)} k_2) \bmod k_0 \,, \\
p_1^{(h+1)} &= (v_0^{(h)} s_1 + p_0^{(h)} + p_1^{(h)} k_3) \bmod k_0 \,, \\
p_2^{(h+1)} &= (v_0^{(h)} s_2 + p_0^{(h)} s_{d+2} + p_1^{(h)} s_{2d+2} + p_{d+1}^{(h)} k_4) \bmod k_0 \,, \\
p_3^{(h+1)} &= (v_0^{(h)} s_3 + p_0^{(h)} s_{d+3} + p_1^{(h)} s_{2d+3} + p_2^{(h)} + p_{d+1}^{(h)} k_5) \bmod k_0 \,, \\
&\begin{cases}
p_4^{(h+1)} = (v_0^{(h)} s_4 + p_0^{(h)} s_{d+4} + p_1^{(h)} s_{2d+4} + p_3^{(h)}) \bmod k_{0,} \\
\vdots \\
p_i^{(h+1)} = (v_0^{(h)} s_i + p_0^{(h)} s_{d+i} + p_1^{(h)} s_{2d+i} + p_{i-1}^{(h)}) \bmod k_0 \,, \\
\vdots \\
p_{d+1}^{(h+1)} = (v_0^{(h)} s_{d+1} + p_0^{(h)} s_{2d+1} + p_1^{(h)} s_{3d+1} + p_d^{(h)}) \bmod k_0 \,,
\end{cases} \\
p_{d+2}^{(h+1)} &= (v_1^{(h)} s_{d+2} + v_2^{(h)} s_{2d+2} + p_{2d+1}^{(h)} k_4) \bmod k_0 \,, \\
p_{d+3}^{(h+1)} &= (v_1^{(h)} s_{d+3} + v_2^{(h)} s_{2d+3} + p_{d+2}^{(h)} + p_{2d+1}^{(h)} k_5) \bmod k_0 \,, \\
&\begin{cases}
p_{d+4}^{(h+1)} = (v_1^{(h)} s_{d+4} + v_2^{(h)} s_{2d+4} + p_{d+3}^{(h)}) \bmod k_0 \,, \\
\vdots \\
p_j^{(h+1)} = (v_1^{(h)} s_j + v_2^{(h)} s_{d+j} + p_{j-1}^{(h)}) \bmod k_0 \,, \\
\vdots \\
p_{2d+1}^{(h+1)} = (v_1^{(h)} s_{2d+1} + v_2^{(h)} s_{3d+1} + p_{2d}^{(h)}) \bmod k_0 \,,
\end{cases} \\
p_{2d+2}^{(h+1)} &= (v_3^{(h)} s_{d+2} + v_4^{(h)} s_{2d+2} + p_{3d+1}^{(h)} k_4) \bmod k_0 \,, \\
p_{2d+3}^{(h+1)} &= (v_3^{(h)} s_{d+3} + v_4^{(h)} s_{2d+3} + p_{2d+2}^{(h)} + p_{3d+1}^{(h)} k_5) \bmod k_0 \,, \\
&\begin{cases}
p_{2d+4}^{(h+1)} = (v_3^{(h)} s_{d+4} + v_4^{(h)} s_{2d+4} + p_{2d+3}^{(h)}) \bmod k_0 \,, \\
\vdots \\
p_k^{(h+1)} = (v_3^{(h)} s_{k-d} + v_4^{(h)} s_k + p_{k-1}^{(h)}) \bmod k_0 \,, \\
\vdots \\
p_{3d+1}^{(h+1)} = (v_3^{(h)} s_{2d+1} + v_4^{(h)} s_{3d+1} + p_{3d}^{(h)}) \bmod k_0 \,,
\end{cases}
\end{aligned}
\tag{3.83}
$$

donde $i = 4, \dots, d+1$, $j = d+4, \dots, 2d+1$ y $k = 2d+4, \dots, 3d+1$. Así como las siguientes expresiones para los valores de las componentes de la 5-tupla $V^{(h+1)}$

$$
\begin{aligned}
v_0^{(h+1)} &= v_0^{(h)} k_1 \,, \\
v_1^{(h+1)} &= v_2^{(h)} k_2 \,, \\
v_2^{(h+1)} &= v_1^{(h)} + v_2^{(h)} k_3 \,, \\
v_3^{(h+1)} &= v_4^{(h)} k_2 \,, \\
v_4^{(h+1)} &= v_3^{(h)} + v_4^{(h)} k_3 \,,
\end{aligned}
\tag{3.84}
$$

para $h \geq 1$, siendo $v_0^{(1)} = k_1$, $v_1^{(1)} = 0$, $v_2^{(1)} = 1$, $v_3^{(1)} = k_2$ y $v_4^{(1)} = k_3$.

Se puede considerar que al desarrollar el producto $M^{h+1} = M^h M$ y simplificar las expresiones, se obtiene una función $F_{(d,K)}: \mathbb{Z}_{k_0}^{6d+7} \to \mathbb{Z}_{k_0}^{3d+5}$, tal que

$$
F_{(d,K)}\left(S, P^{(h)}, V^{(h)}\right) = \left(f_{(d,K)}^{h}(S), V^{(h+1)}\right) = \left(P^{(h+1)}, V^{(h+1)}\right). \tag{3.85}
$$

Universitat d'Alacant
Universidad de Alicante

Con el enfoque optimizado, el número de operaciones para calcular una iteración, en lugar de ser de orden cúbico respecto a d (calculando la potencia de una matriz) es de orden lineal (al calcular simplemente una tupla).

Se dispone ahora de un algoritmo basado en las propiedades de potencias de matrices TSB de 3×3 bloques cuya implementación simplemente consiste en recorrer las componentes de un vector y aplicarles una serie de operaciones preestablecidas.

3.5.2 Resultados

Las pruebas realizadas con este enfoque han proporcionado magníficos resultados. En una primera implementación *ad hoc* para $d = 7$ y $d = 19$ se ha conseguido multiplicar su velocidad por 4 y 5 respectivamente, resultando ambos generadores los más rápidos en la comparativa con los algoritmos AES (OFB), RC4, Salsa20 y HC128, como se muestra en la tabla 3.11.

AES	RC4	Salsa20	HC128	M7	M19
60 MB/s	154 MB/s	138 MB/s	131 MB/s	244 MB/s	260 MB/s

Tabla 3.11: Comparativa con el enfoque optimizado para $d = 7$ y $d = 19$

Las velocidades que aparecen en la tabla para los dos generadores (M7 y M19) se han calculado en las mismas condiciones de ejecución que para las versiones anteriores (tabla 3.9), es decir, a partir de varias secuencias de un gigabyte de longitud (2^{33} bits) obtenidas sobre un procesador Intel i7 920 de 2,67 GHz con Windows 8.1. Cada secuencia de 1 GB tarda en generarse (evitando su escritura sobre el sistema de ficheros) aproximadamente 4 segundos, con sólo unas centésimas de segundo de diferencia entre unas pruebas y otras.

Para que la implementación resulte óptima, el código implementado ya no es parametrizable en tiempo de ejecución, compilándose un código distinto para cada caso (M7 para $d = 7$ y M19 para $d = 19$). Además, la implementación *ad hoc* no utiliza un bucle para recorrer las tuplas y calcular sus componentes: los cálculos se han codificado, simplemente, como una secuencia de operaciones.

Con el objetivo de facilitar las pruebas con otros valores de d, se ha incorporado a la implementación el cálculo de las componentes de las tuplas mediante bucles controlados por el valor de d, y se posibilita, al menos, la parametrización del código en tiempo de compilación. De esta forma, estableciendo el valor de d antes de compilar, se obtiene el código apropiado respecto a los valores de la tupla K y al número de componentes del resto de tuplas.

Esta implementación se ha probado con los dos valores extremos de d usados previamente, y con dos nuevos valores intermedios ($d = 10$ para M10 y $d = 14$ para M14). Las velocidades obtenidas (esta vez entre 100 semillas, en lugar de entre 10) se muestran en la tabla 3.12. En la versión *ad hoc*, el valor de escogido de d afectaba de forma más sensible al tiempo de computación, mientras que la versión con un bucle genérico mantiene velocidades muy similares para distintos valores de d, aumentando ligeramente la velocidad obtenida en M7 y disminuyendo algo la de M19.

M7	M10	M14	M19
248 MB/s	250 MB/s	244 MB/s	249 MB/s

Con mejora en filtrado

M10	M14
290 MB/s	278 MB/s

Tabla 3.12: Comparativa para mejora en d par

La elección para las pruebas de los valores intermedios 10 y 14 no es casual, ya que posteriormente se ha probado una versión de esos dos generadores (M10 y M14) incorporando una mejora en el filtrado que es más fácil de aplicar a los valores pares de d.

Si de cada elemento de una (3d+2)-tupla $P^{(h)}$ se extraen 2 bytes, en cada iteración se obtienen $6d + 4$ bytes. El filtrado *s-box* 32×32 se realiza de 4 en 4 bytes, y hasta ahora se ha aplicado después de la extracción de dos iteraciones. La razón para usar estas extracciones dobles (de $12d + 8$ bytes) es para garantizar que el filtrado se aplica a un número de bytes múltiplo de 4 independientemente del valor de d, simplificando así la implementación genérica (para cualquier d) del filtrado.

Aplicar el filtrado *a posteriori* de la extracción supone recorrer dos veces el buffer usado para el almacenamiento de los bytes extraídos. La mejora indicada en la tabla 3.12 se refiere a filtrar la secuencia conforme va siendo extraída (de 4 en 4 bytes), sin recorrer una segunda vez el buffer.

Como ya se ha comentado, esta estrategia resulta más fácil de implementar para valores pares de d que proporcionan directamente un múltiplo de 4 bytes en una sola iteración, pero una vez comprobada su rentabilidad para los casos M10 y M14, se ha incorporado para todos los valores de d al código que finalmente se presenta en este trabajo como resultado óptimo (apéndice B.3).

La tabla 3.13 muestra tiempos y velocidades para generadores de distintos tamaños de d con la implementación óptima. En ella se aprecia que la velocidad es prácticamente la misma en todos los casos, próxima los 300 MB/s, compensando la rapidez en el cálculo de cada iteración en el caso de los valores más pequeños de d, con la productividad por iteración en el caso de los valores más grandes de d.

	d	s/GB	Bytes/Ite.	MB/s	Iteraciones/GB	s/Iteración
M7	7	3,58	46	286,2	23342213,57	1,53E-07
M8	8	3,52	52	291,3	20648881,23	1,70E-07
M9	9	3,44	58	297,3	18512790,07	1,86E-07
M10	10	3,52	64	291,3	16777216	2,10E-07
M11	11	3,48	70	294,6	15339168,91	2,27E-07
M12	12	3,52	76	291,1	14128181,89	2,49E-07
M13	13	3,63	82	281,7	13094412,49	2,78E-07
M14	14	3,68	88	277,9	12201611,64	3,02E-07
M15	15	3,55	94	288,8	11422785,36	3,10E-07
M19	19	3,51	118	291,4	9099506,983	3,86E-07

Tabla 3.13: Velocidades y tiempos en implementación óptima

De nuevo, los tiempos se han obtenido generando 10 secuencias distintas de un gigabyte cada una, y en el mismo entorno que en anteriores ocasiones.

El generador M10 tiene como característica especial que genera 64 bytes por iteración (512 bits), es decir 2^6 bytes (2^9 bits). El hecho de producir un número de bits potencia de 2 por iteración lo convierte en ideal para trabajar en registros y buffers de tamaños habituales en la lógica binaria usada en computación.

Como última fase de experimentación, y una vez obtenido un tipo de generador suficientemente rápido y competitivo, se han probado los más exhaustivos conjuntos de TestU01 (véase la sección 1.3.4.2).

Se ha conseguido pasar todos los conjuntos de test de TestU01 con los generadores M7, M10, y M19, desde la Small Crush de 15 test, pasando por Alfabit de 17 test, Rabbit de 33, Crush de 144, y hasta llegar a la más completa, Big Crush de 160 test, en este último caso, analizando secuencias próximas al gigabyte. En el apéndice A.4 se muestra un informe exhaustivo del conjunto Big Crush con todos los test favorables para el generador M10. Los análisis TestU01 con cada uno de los cinco conjuntos de test se han repetido para 10 distintas secuencias usando identificadores de clave distintos que seleccionan semillas aleatorias distintas, obteniéndose resultados estadísticos muy satisfactorios que se pueden comprobar en la tabla de test superados que se muestra también en el apéndice A.4, o consultando los informes en el enlace que parece en el mismo apéndice.

Universitat d'Alacant
Universidad de Alicante

4 Conclusiones

4.1 El generador propuesto

El objetivo principal de este trabajo ha sido el diseño de un generador pseudoaleatorio basado en matrices triangulares superiores de 3×3 bloques adecuado para ser empleado como generador de secuencia cifrante en un criptosistema de cifrado en flujo binario aditivo.

Se ha implementado un generador configurable para distintos tamaños del tercer bloque de la diagonal, proporcionando, para cualquiera de los tamaños probados, una velocidad en la generación de bits competitiva con respecto a otros algoritmos vigentes con los que se ha comparado.

El tamaño 10×10 escogido para el tercer bloque cumple con los niveles de aleatoriedad que se consideran más estrictos en la actualidad (PracRand y TestU01). La elección de este valor proporciona un generador con menor latencia que si se escogen otros valores de d superiores. En concreto, M10, proporciona 64 bytes por iteración, un número apropiado que resulta ideal para su implementación en múltiples arquitecturas, al tratarse de una potencia de dos (2^6).

Aunque finalmente se ha propuesto un generador especifico con una parametrización concreta, se debe enfatizar que en realidad se trata de una familia de generadores que se pueden adecuar, sin pérdida de aleatoriedad ni aumento de coste computacional apreciable, a distintos requisitos impuestos por la aplicación, implementación, arquitectura, etc., con sólo variar el parámetro d de la versión optimizada.

Universitat d'Alacant
Universidad de Alicante

4.1.1 Posibilidades de uso

Dado el rendimiento alcanzado por el generador optimizado finalmente propuesto, resulta perfectamente adecuado para su uso en protocolos, sistemas y aplicaciones que requieran de criptografía simétrica. Además, considerando algunas características de seguridad adicionales, como el soporte directo de tamaños de clave variable, puede ser interesante como sustituto de otros algoritmos que han sido considerados inseguros recientemente, como RC4.

El generador propuesto tiene la característica de proporcionar una extracción de 64 bytes por iteración, que resulta apropiada para su implementación tanto en software como en hardware o, incluso, en hardware reconfigurable (FPGA). Por ejemplo, en arquitecturas de 64 bits, proporcionaría 8 registros de 64 bits, que pueden ser interpretados como una matriz de 8×8 bytes. La memoria necesaria para almacenar la *s-box* de 32×32 (4 cajas de 8×32) sería de 4 KBytes y para las tuplas de constantes, semilla, estado interno y variables auxiliares haría falta algo menos de 2 KBytes adicionales.

La variación del valor de d afecta a la latencia del generador, que para su uso en cifrado en flujo es conveniente que sea baja. Al reducir el valor de d se reducen los bpi y, con ello, la latencia (posibles retardos en el flujo ante la demanda de un nuevo bit); además del consumo de otros recursos como la memoria.

Universitat d'Alacant
Universidad de Alicante

4.2 Contribuciones

4.2.1 Extensión a 3×3 bloques

El generador propuesto parte de un estudio previo basado en las potencias de las matrices triangulares superiores por bloques (TSB) de 2×2 bloques definidas en $\mathbb{Z}_p$, con p primo. Se ha extendido la formulación de las matrices TSB de 2×2 bloques a un planteamiento multibloque, obteniendo las expresiones para matrices TSB de 3×3 bloques, que constituyen la base matemática del generador propuesto.

El uso de matrices TSB de 3×3 bloques, respecto a las de 2×2 bloques, permite, ajustando una serie de parámetros y métodos (extracción y filtrado), aumentar el periodo y conseguir mayor rendimiento con unas muy buenas propiedades en términos de aleatoriedad.

4.2.2 Estudio y parametrización

Para ajustar los parámetros al generador se ha realizado una estratégica y metódica búsqueda en un extenso ámbito de posibles combinaciones, consistente en diseñar y analizar varias baterías de pruebas que han permitido acotar, cada vez más, la búsqueda y dirigir ésta hacia las combinaciones con mejores resultados estadísticos.

Finalmente, la búsqueda se ha centrado en aquellas combinaciones que cumplen con unos requisitos mínimos establecidos para garantizar un adecuado nivel de seguridad criptográfica, definidas como combinaciones admisibles.

4.2.3 Bloques de tamaño 1×1

La inclusión de combinaciones con bloques 1×1 ha resultado una aportación novedosa respecto a anteriores trabajos con matrices TSB. Estos bloques permiten alcanzar periodos grandes con muy poco cálculo y sin perjuicio en términos de aleatoriedad.

4.2.4 Extracción con aprovechamiento máximo

Se ha concluido que los primos de mayor tamaño aportan mayor aleatoriedad y, además, proporcionan los mejores rendimientos si se escogen y explotan con métodos de extracción adecuados. Se ha buscado primos p tales que los elementos de $\mathbb{Z}_p$ contengan el máximo de bits equiprobables para los valores 0 y 1; con el objetivo de aprovechar esos bits en el sistema de extracción, y maximizar los bpi. Los primos próximos a 2^n son los mejores candidatos para maximizar este criterio.

Universidad de Alicante

4.2.5 Tablas de polinomios primitivos

La selección de primos y de tamaños de bloques está limitada por la necesidad de conocer un polinomio primitivo que permita la construcción del bloque. Por ello, se ha realizado, también, una búsqueda de estos polinomios que ha condicionado la elección de primo y tamaño de bloque.

4.2.6 Inclusión de filtrado con *s-box* 32×32

Un filtrado con cajas de sustitución de 32×32 bits ha permitido, mediante su propio proceso de inicialización, incorporar al generador el concepto de clave. A partir de la misma se determina la configuración concreta del filtro y se

obtiene una semilla adecuada, eliminando la posibilidad de semillas no válidas o nulas y dificultando el criptoanálisis. Éste filtrado es balanceado (no altera las características estadísticas) y altamente no lineal (evitando ataques basados en el criptoanálisis lineal o diferencial).

4.2.7 Algoritmo optimizado

Se ha llegado a la conclusión de que, para el generador propuesto, se debe trabajar con el primo 65537 y bloques de tamaños 1×1 y 2×2. El estudio sobre el tamaño del tercer bloque ha estado delimitado a los polinomios primitivos de grado 7 o superior encontrados para ese primo (restricción impuesta a las combinaciones admisibles).

La forma de los polinomios primitivos hallados (trinomios mónicos con término independiente y término en x no nulos) ha permitido replantear de una forma mucho más eficiente el algoritmo, haciéndolo específico para el subconjunto de matrices TSB que se pueden construir con ese tipo de polinomios primitivos.

4.2.8 Publicaciones

El generador descrito ha sido sometido, durante distintas fases de su desarrollo, al análisis de la comunidad científica en dos ocasiones, con las siguientes publicaciones "Cryptographic Applications of 3×3 Block Upper Triangular Matrices" [15], y "Extensión y parametrización de un generador pseudoaleatorio matricial" [14]. Además, durante su periodo de investigación se han producido otras publicaciones relacionadas [16, 17].

4.3 Propuestas futuras

4.3.1 Otros primos

El mínimo valor de d se ha establecido en 7 debido a la restricción impuesta al periodo (128 mbp). Una propuesta para el futuro consiste en utilizar primos mayores que permitan reducir el valor de d hasta 3 y alcanzar los tamaños de bloque mínimos en la diagonal de M; 1×1, 2×2 y 3×3. Como se ha comprobado, la falta de una equiprobabilidad estricta para los bits menos significativos (bms) de los elementos de $\mathbb{Z}_p$ usados en la extracción no afecta a los test de aleatoriedad

de forma significativa. Cabe esperar, pues, que aunque el primo escogido no sea tan próximo a una potencia de 2 como lo es 65537, se puedan aprovechar con otros primos igualmente, y sin problemas, todos los bits menos uno (el más significativo) de los elementos de $\mathbb{Z}_p$.

Así, por ejemplo, con $p = 536870923$, en una matriz con bloques en la diagonal de tamaño 1×1, 2×2 y 3×3, el periodo es de 139 mbp, y si se extraen 28 bms de los 29 bits que tiene cada uno de los 11 elementos de $\mathbb{Z}_p$ disponibles para la extracción, se obtienen 308 bpi. Para realizar las operaciones previstas entre elementos de $\mathbb{Z}_p$ con el enfoque optimizado (la más restrictiva respecto a tamaño de registro es la suma de cinco productos) se necesitan 61 bits ($\log_2(5p^2)$), por lo que para $p = 536870923$ se podría seguir usando una implementación con enteros de 64 bits.

4.3.2 Implementaciones específicas

Ya se ha comentado la posibilidad de explotar el generador propuesto con $d = 10$ en una implementación hardward dadas las distintas posibilidades de interpretar los 64 bytes obtenidos por iteración (32 registros de 16 bits, 16 de 32 bits o 8 de 64 bits).

El algoritmo permite su paralelización, tanto en una propuesta hardware, como en otro tipo de implementación software. Para una implementación concurrente, algunas de las estrategias más comunes en la actualidad se basan en el uso GPU's mediante técnicas GPGPU (*General Purpose Computing on Graphics Processing Units*) o también en el uso de las instrucciones AVX (*Advanced Vector Extensions*) o PNI (*Prescott New Instructions*) de los microprocesadores x86 (Intel llamó Prescott a su juego de instrucciones SSE3).

En microprocesadores x86, las instrucciones SSE (*Streaming* SIMD *Extensions*) permiten operar con enteros de 128 bits mediante técnicas SIMD (*Single Instruction, Multiple Data*) de paralelismo a nivel de datos. Una implementación (total o parcial) en ensamblador con estas instrucciones permitiría trabajar con mayores primos aunque necesitasen enteros de 128 bits para operar.

4.3.3 Nuevos planteamientos con 2×2 bloques

Con primos suficientemente grandes, cabría replantearse volver al esquema usado con anterioridad

$$M = \begin{bmatrix} A & X \\ 0 & B \end{bmatrix} \in \Theta_2 \,, \tag{4.86}$$

siendo A un bloque de tamaño 1×1 y B de tamaño $d \times d$, y aplicar el enfoque de optimización planteado en este trabajo cuando se usan trinomios primitivos con término en x no nulo; siempre y cuando el tamaño del primo y d garanticen un periodo y un espacio de claves lo suficientemente grandes.

4.3.4 Búsquedas de polinomios primitivos

Hay que tener en cuenta que cualquier proyecto futuro que pretenda usar primos más grandes se encontrará con una primera y necesaria fase de búsqueda de polinomios primitivos. Esta búsqueda es una ardua tarea con la entidad suficiente para constituir otro proyecto en sí misma.

4.3.5 Extensión a más bloques

Más allá de aumentar el tamaño del primo, otro futuro planteamiento consiste en continuar con la extensión a múltiples bloques con el estudio de Θ_4; demostrando un nuevo teorema como el teorema 2.2 o el teorema 2.3 del que se obtengan expresiones similares a las (2.31), (2.32) y (2.41), (2.42), (2.43) para una matriz

$$M = \begin{bmatrix} A & X_{1,2} & X_{1,3} & X_{1,4} \\ 0 & B & X_{2,3} & X_{2,4} \\ 0 & 0 & C & X_{3,4} \\ 0 & 0 & 0 & D \end{bmatrix}. \tag{4.87}$$

Universitat d'Alacant
Universidad de Alicante

Se podría, entonces, calcular también la correspondiente optimización para el caso de una diagonal con bloques de tamaños $1 \times 1, 2 \times 2, 3 \times 3, \ d \times d$ creados, también, mediante trinomios con coeficiente en x no nulo.

Por ejemplo, con $p = 65537$, este nuevo planteamiento de 4×4 bloques necesitaría de $d = 5$ para alcanzar un periodo de 127 mbp y 82 bytes por iteración.

Otra propuesta, con mayor grado de complejidad, consiste en generalizar los teoremas y recurrencias para Θ_n.

4.3.6 Criptoanálisis

Una última propuesta, en el marco del criptoanálisis, consiste en el estudio en mayor profundidad de la seguridad del criptosistema propuesto; trabajo que, no obstante, queda abierto a la comunidad científica.

Bibliografía

[1] Aguirre, J., Alvarez, R., Tortosa, L., & Zamora, A. (2007). Fast Pseudorandom Generator Based on Packed Matrices. *Avanced Topics in Information Security and Privacy, 101 (ISSN 1790-5117), 98.*

[2] Aguirre, J., Alvarez, R., Tortosa, L., & Zamora, A. (2007). Generador pseudoaleatorio matricial optimizado sobre $\mathbb{Z}_2$. *II Simposio sobre Seguridad Informática.*

[3] Aguirre, J., Alvarez, R., Tortosa, L., & Zamora, A. (2008). An Optimized Pseudorandom Generrator Using Packed Matrices. *Transactions on Information Science Applications, 5* (ISSN 1790-0832), 487-496.

[4] Alvarez, G., Montoya, F., & Peinado, A. (1998). Generación de Claves del Criptosistema de Clave Pública de Blum, Blum y Shub. *Proc. V Reunión Española sobre Criptología*, 55-65.

[5] Alvarez, R. (2005). *Aplicaciones de las matrices por bloques a los criptosistemas de cifrado en flujo.* Ph.D. dissertation, Universidad de Alicante.

[6] Alvarez, R., & McGuire, G. (2009). S-Boxes, APN Functions and Related Codes. *Enhancing Cryptographic Primitives with Techniques from Error Correcting Codes, 23*, 49-62.

[7] Alvarez, R., & Zamora, A. (2014). A Matrix PRNG with S-Box Output Filtering. *Journal of Applied Mathematics, 2014.*

[8] Alvarez, R., & Zamora, A. (2014). Randomness Analysis of Key-Derived S-Boxes. *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, (págs. 611-618).

[9] Alvarez, R., Castel, M. J., Tortosa, L., & Zamora, A. (2009). Optimizing matrix operations in Z2 by word packing. *Applied Mathematics Letters, 22* (2), 242-244.

[10] Alvarez, R., Climent, J. J., Tortosa, L., & Zamora, A. (2003). A Pseudorandom Bit Generator Based on Block Upper Triangular Matrices. En *Web*

Universitat d'Alacant
Universidad de Alicante

Engineering: Internacional Conference, ICWE 2003 LNCS 2722 (299-300). Springer.

[11] Alvarez, R., Climent, J. J., Tortosa, L., & Zamora, A. (2004). Un generador matricial de claves frente a Blum Blum Shub. *Avances en criptología y seguridad de la información*, 113-123.

[12] Alvarez, R., Climent, J., Tortosa, L., & Zamora, A. (2005). An efficient binary sequence generator with cryptographic applications. *Applied Mathematics and Computation, 167*, 16-27.

[13] Alvarez, R., Ferrandez, F., Vicent, J., & Zamora, A. (2006). Applying quick exponentiation for block upper triangular matrices. *Applied Mathematics and Computation, 183*, 729-737.

[14] Alvarez, R., Marti}nez, F.-M., Vicent, J.-F., & Zamora, A. (2012). Extensión y parametrización de un generador pseudoaleatorio matricial. *Actas de la XII Reunión Española sobre Criptología y Seguridad de la Información*.

[15] Alvarez, R., Martinez, F. M., Vicent, J., & Zamora, A. (2012). Cryptographic Applications of 3 × 3 Block Upper Triangular Matrices. En L. N. in Computer Science (Ed.), *Hybrid Artificial Intelligent Systems* (págs. 97-104). Springer.

[16] Alvarez, R., Martinez, F.-M., Vicent, J.-F., & Zamora, A. (2007). A New Public Key Cryptosystem Based on Matrices. *6th WSEAS International Conference on Information Security and Privacy, Tenerife, Spain, December 14-16. 200*, págs. 36-39. Advanced Topics in Information Security and Privacy.

[17] Alvarez, R., Martinez, F.-M., Vicent, J.-F., & Zamora, A. (2008). A Matricial Public Key Cryptosystem with Digital Signature. *WSEAS Transactions on Mathematics, 1*, 1.

[18] Alvarez, R., Tortosa, L., Vicent, J. F., & Zamora, A. (2004). An Integral Security Kernel. *WSEAS Transactions on Business and Economics*, 241-246.

[19] Alvarez, R., Tortosa, L., Vicent, J., & Zamora, A. (2009). A non-abelian group based on block upper triangular matrices with cryptographic applications. En L. N. in Computer Science 5527 (Ed.), *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* (págs. 117-126). Springer.

[20] Anderson, R. (1995). On Fibonacci Keystream Generators. *Fast Software Encryption*, (págs. 346-352).

[21] Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., & Rechberger, C. (2008). New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. En K. Nyberg (Ed.), *Fast Software Encryption* (Vol. 5086, págs. 470-488). Springer Berlin Heidelberg. Obtenido de *http://dx.doi.org/10.1007/978-3-540-71039-4_30*

[22] Babbage, S., Canniere, C., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., ... Robshaw, M. (2008). The eSTREAM Portfolio. *eSTREAM, ECRYPT Stream Cipher Project*.

Universitat d'Alacant
Universidad de Alicante

[23] Beker, H., & Piper, F. C. (1982). *Cipher Systems: The Protection of Communications*. New York: John Wiley and Sons.

[24] Berlekamp, E. R. (1968). *Algebraic Coding Theory*. New York: McGraw Hill.

[25] Bernstein, D. J. (2008). ChaCha, a Variant of Salsa20. *Workshop Record of SASC*, *8*.

[26] Bernstein, D. J. (2008). Notes on the ECRYPT Stream Cipher Project (eSTREAM). *http://cr.yp.to/streamciphers.html*.

[27] Bernstein, D. J. (2008). The Salsa20 Family of Stream Ciphers. En *New stream cipher designs* (págs. 84-97). Springer.

[28] Blum, L., Blum, M., & Shub, M. (1986). A Simple Unpredictable Pseudorandom Number Generator. *SIAM Journal on Computing, 15-2*, 364-383.

[29] Boyar, J. (1989). Inferring Sequences Generated by a Linear Congruential Generator Missing Low-Order Bits. *Journal of Cryptology, 1*, 177-184.

[30] Brennab, J. J., & Geist, B. (1998). Analysis of Iterated Modular Exponentiation: The Orbits of x^a mod n. *Designs, Codes and Cryptography, 13*, 229-245.

[31] Bruer, J. O. (1984). *On Pseudorandom Sequences as Crypto Generators*. Switzerland: Proc. Int. Zurich Seminar on Digital Communication.

[32] Caballero-Gil, P., & Fuster-Sabater, A. (2005). Improvement of the Edit Distance Attack to Clock-Controlled LFSR-Based Stream Ciphers. En *Computer Aided Systems Theory--EUROCAST 2005* (págs. 355-364). Springer.

[33] Campbell, C. M. (1978). Design and Specification of Cryptographic Capabilities. *IEEE Computer Society Mag., 16*, 15-19.

[34] Campbell, J., Easter, R., Lee, A., & Snouffer, R. (2003). Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules. *NIST. FIPS PUB*, 140-2.

[35] Chabuad, F. (June de 2000). The Primitive and Irreducible Polynomial Server (PIPS). *The Primitive and Irreducible Polynomial Server (PIPS)*. Obtenido de *http://zenfact.sourceforge.net/defaultind_COUNT_1_FILE0_PIPS.html*

[36] Chabuad, F., & Lercier, R. (June de 2000). ZEN. *ZEN*. Obtenido de *http://zenfact.sourceforge.net*

[37] Cusick, T. W. (1995). Properties of the Pseudorandom Number Generator. *IEEE Trans. Information Theory, 41*, 1155-1159.

[38] Daeman, J., Govaerts, R., & Vandewalle, J. (1994). Resynchronization Weaknesses in Synchonous Stream Ciphers. En *93, Springer-Verlag* (págs. 159-167). Proc. EUROCRYPT.

Universitat d'Alacant
Universidad de Alicante

[39] Daemen, J., & Rijmen, V. (1999). AES Proposal: Rijndael. *1st Advanced Encryption Standard (AES1) Conference*, (págs. 343-348).

[40] Davies, D. W., & Parkin, G. I. (1983). The Average Size of the Key Stream in Output Feedback Encipherment. En *Proc* (págs. 263-279). Springer-Verlag: Workshop in Cryptography.

[41] Doty-Humphrey, C. (November de 2014). Practically Random. *PractRand 0.92*. Obtenido de *http://sourceforge.net/projects/pracrand/*

[42] Eastlake, D., Crocker, S., & Schiller, J. (1994). Randomness Recommendations for Security. *Network Working Group, RFC, 1750*.

[43] Fuster-Sabater, A. (2014). Design of Nonlinear Filters with Guaranteed Lower Bounds on Sequence Complexity. *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, (págs. 557-566).

[44] Golomb, S. W. (1982). *Shift register sequences*. Aegean Park Press.

[45] Guanella, G. M. (1946). Means for and Method for Secret Signalling. *U.S. Patent, 2405400*.

[46] Gupta, S. S., Maitra, S., Paul, G., & Sarkar, S. (2014). (Non-) Random Sequences from (Non-) Random Permutations—Analysis of RC4 Stream Cipher. *Journal of cryptology, 27* (1), 67-108.

[47] Hernandez, L., Muñoz, F., Montoya, G., & Peinado, A. (1998). Maximal Period of Orbits of the BBS Generator. *Proc. CISC ', 98*, 71-80.

[48] Hogg, R. V., & Tannis, E. A. (1988). *Probability and Statistical Inference*. New York: Macmillan Publishing.

[49] Hussain, I., Shah, T., Gondal, M. A., & Khan, W. A. (2011). Construction of Cryptographically Strong 8 × 8 S-boxes. *World Applied Sciences Journal, 13* (11), 2389-2395.

[50] Jueneman, R. R. (1983). Analysis of Certain Aspects of Output-Feedback Mode. *Proc. CRYPTO ', 82*, 99-127.

[51] Kelsey, J., Schneier, B., & Ferguson, N. Y. (1999). Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. En *99, LNCS 1758, Springer Verlag* (págs. 13-33). Proc. Selected Areas in Cryptography.

[52] Klein, A. (2008). Attacks on the RC4 Stream Cipher. *Designs, Codes and Cryptography, 48*(3), 269-286.

[53] Knudsen, L. (1998). Block Ciphers - A Survey. En *LNCS 1528* (págs. 18-48). Springer-Verlag: State of the Art in Applied Cryptography.

[54] Knuth, D. E. (1981). *The Art of Computer Programming - Seminumerical Algorithms Volume 2*. Addison-Wesley.

Universitat d'Alacant
Universidad de Alicante

[55] Krawczyk, H. (1992). How To Predict Congruential Generators. *Journal of Algorithms, 13*, 527-545.

[56] Kuhn, G. J. (1988). Algorithms for Self-Synchronizing Ciphers. *Proc. COMSIG ', 88.*

[57] Lagarias, J. C. (1990). Pseudorandom Number Generators in Cryptography and Number Theory. *Proc. Symposia in Applied Mathematics, 42*, 115-143.

[58] L'Ecuyer, P. (August de 2009). *TestU01.* 1.2.3. Obtenido de *http://simul.iro.umontreal.ca/testu01/tu01.html*

[59] L'Ecuyer, P., & Simard, R. (2007). TestU01: AC Library for Empirical Testing of Random Number Generators. *ACM Transactions on Mathematical Software (TOMS), 33*(4), 22.

[60] Lehmer, D. (1951). *Mathematical methods in large-scale computing units.* Proceedings of a Second Symposium on Large Scale Digital Calculating Machinery.

[61] Lidl, R., & Niederreiter, H. (1983). Finite Fields: Encyclopedia of Mathematics and Its Applications. *Computers and Mathematics with Applications, 20*. Addison-Wesley.

[62] Lidl, R., & Niederreiter, H. (1994). *Introduction to Finite Fields and their Applications.* Cambridge: Cambridge University Press.

[63] Luby, M. (1996). *Pseudorandomness and Cryptographic Applications.* Princeton University Press.

[64] Lucena-Lopez, M.-J. (2015). *Criptografía y seguridad en computadores* 4-0.11.0. Obtenido de *http://criptografiayseguridad.blogspot.com.es/*

[65] Massey, J. L.-R., & Decoding, B. (1969). IEEE Transactions of Information Theory. *vol., 15*, 122-127.

[66] Mathews, T. (1996). Suggestions for Random Number Generation in Software. *News and Advices from RSA Laboratories, Bulletin, 1.*

[67] Maurer, U. (1995). The Role of Information Theory in Cryptography. *Cryptography and Coding IV, Inst. of Mathematics and its Applications*, 49-71.

[68] Menezes, A., van Oorschot, P., & Vanstone, S. (2001). *Handbook of Applied Cryptography.* Florida: CRC Press.

[69] Mes, J. O. (april de 2015). Dev-C++ *Orwell* Obtenido de *http://orwelldevcpp.blogspot.com*

[70] Mister, S., & Tavares, S. (1998). Cryptanalysis of RC4-like Ciphers. *Select Areas In Cryptography, LNCS, 1556*, 131-143.

[71] Murphy, S., & Robshaw, M. (2002). Key-Dependent S-Boxes and Differential Cryptanalysis. *Designs, Codes and Cryptography, 27*(3), 229-255. Obtenido de *http://dx.doi.org/10.1023/A%3A1019991004496*

[72] National Bureau of Standards. (1980). DES Modes of Operation. *Federal Information Processing Standards Publication (FIPS PUB) 81.*

[73] NIST (2000). A Statistical Test Suit for Random and Pseudorandom Number Generatiors for Cryptographic Applications Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *NIST Special Publication*, 800-22.

[74] O'Neill, M. E. (2015). Empirical Statistical Tests. Obtenido de *http://www.pcg-random.org/statistical-tests.html*

[75] O'Connor, S. E. (1999-2015). COMPUTING PRIMITIVE POLYNOMIALS *Primpoly Version 12.0*. Obtenido de *http://www.seanerikoconnor.freeservers.com* /Mathematics/AbstractAlgebra/PrimitivePolynomials/overview.html

[76] Odoni, R. W., Varadharajan, V., & Sanders, P. W. (1984). Public Key Distribution in Matrix Rings. *Electronic Letters, 20*(9), 386-387.

[77] Peinado, A., & Fuster-Sabater, A. (2013). Generation of Pseudorandom Binary Sequences by Means of Linear Feedback Shift Registers (LFSRs) with Dynamic Feedback. *Mathematical and Computer Modelling, 57*(11), 2596-2604.

[78] Plumstead, J. B. (1983). Inferring a Sequence Generated by a Linear Congruence. *Proc. CRYPTO ', 82*, 317-319.

[79] Preneel, B., Nuttin, M., Rijmen, V., & Buelens, J. (1994). *Cryptanalysis of the CFB mode of the DES with a Reduced Number of Rounds.* Springer-Verlag 212--223: Proc. CRYPTO '93.

[80] Rifa, J., & Huguet, L. (1991). *Comunicación Digital.* Masson.

[81] Rivest, R. L. (1992). *The RC4 Encryption Algorithm.* Inc: RSA Data Security.

[82] Robshaw, M. (2008). The eSTREAM Project. En *New Stream Cipher Designs* (págs. 1-6). Springer.

[83] Robshaw, M. J. (1994). *Technical Report TR-401.* RSA Laboratories.

[84] Robshaw, M. J. (1995). *Technical Report TR-701 2.0.* RSA Laboratories.

[85] Rueppel, R. A. (1986). *Analysis and Design of Stream Ciphers.* Springer-Verlag.

[86] Rueppel, R. A. (1987). *When Shift Registers Clock Themselves.* Springer-Verlag 53--64: Proc. EUROCRYPT '87.

[87] Rueppel, R. A. (1992). Stream Ciphers. Contemporary Cryptology: The Science of Information Integrity, G. *J. Simmons, IEEE press*, 65-134.

Universitat d'Alacant
Universidad de Alicante

[88] Savage, J. E. (1967). Some Simple Self-Synchronizing Digital Data Scramblers. *Bell System Technical Journal, 46*, 448-487.

[89] Schneier, B. (2015). *Applied Cryptography: Protocols, Algorithms and.* (S. C. in C (30th Anniversary ed.). New York: Wiley Computer Publishing, John Wiley & Sons, Inc.

[90] Selmer, E. S. (1966). *Linear Recurrence over Finite Fields.* Norway: University of Bergen.

[91] Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal, 27*(379), 623-656.

[92] Shannon, C. E., & Weaver, W. (1949). *The Mathematical Theory of Communication.* University of Illinois press.

[93] Smeets, B. (1986). *A Note on Sequences Generated by Clock-Controlled Shift Registers.* Springer-Verlag 40--42: Proc. EUROCRYPT '85.

[94] Stallings, W. (2014). *Cryptography and Network Security: Principles and Practice* (Sixth ed.). New Jersey: Prentice Hall.

[95] Vazirani, U. V., & Vazirani, V. V. (1985). Efficient and Secure Pseudorandom Number Generation. *Advances in cryptology*, (págs. 193-202).

[96] Vicent-Frances, J.-F. (2007). *Propuesta y análisis de criptosistemas de clave pública basados en matrices triangulares superiores por bloques.* Ph.D. dissertation, Universidad de Alicante.

[97] Wackerly, D., Mendenhall, W., & Scheaffer, R. (1996). *Mathematical Statistics with Applications.* Duxbury Press.

[98] Wu, H. (2004). A New Stream Cipher HC-256. En B. Roy, & W. Meier (Edits.), *Fast Software Encryption* (Vol. 3017, págs. 226-244). Springer Berlin Heidelberg. Obtenido de *http://dx.doi.org/10.1007/978-3-540-25937-4_15*

[99] Wu, H. (2008). The Stream Cipher HC-128. En *New Stream Cipher Designs* (págs. 39-47). Springer.

[100] Yao, A. C. (1982). and Applications of Trapdoor Functions. *Proc* (págs. 80-91). 23rd IEEE Symposium on Foundations of Computer Science.

[101] Zierler, N. (1959). Linear Recurring Sequences. *Journal of the Society for Industrial and Applied Mathematics, 7*(1), 31-48

Universitat d'Alacant
Universidad de Alicante

Índices y acrónimos

Índice de tablas

Tabla 1.1: Tabla de trasformaciones en una ronda doble de Salsa20 30
Tabla 3.1: Ejemplo de combinaciones seleccionables 51
Tabla 3.2: Combinaciones *top*, *min* y *bottom* 52
Tabla 3.3: Probabilidades al bit 57
Tabla 3.4: Combinaciones probadas..... 59
Tabla 3.5: Mínimo de bits para el periodo y la semilla por combinación 60
Tabla 3.6: Bits por iteración y número de iteraciones por combinación..... 61
Tabla 3.7: Estimación de los costes por iteración y por bit 62
Tabla 3.8: Combinaciones admisibles probadas para 65537 68
Tabla 3.9: Comparativa velocidades combinaciones 1-2-7 y 1-2-19..... 69
Tabla 3.10: Valores de $k4$ y $k5$ para distintos valores de d..... 71
Tabla 3.11: Comparativa con el enfoque optimizado para $d = 7$ y $d = 19$ 74
Tabla 3.12: Comparativa para mejora en d par 75
Tabla 3.13: Velocidades y tiempos en implementación óptima 76

Universidad de Alicante

Índice de figuras

Figura 1.1: Esquema original de comunicación de Shannon 5
Figura 1.2: Forma de Fibonacci para un LFSR 13
Figura 1.3: Forma de Galois para un LFSR 15
Figura 1.4: Cifrado de Vernam 23
Figura 1.5: Cifrado en modo *output feedback* (OFB) 27
Figura 1.6: Esquema de funcionamiento del PRGA de RC4 28
Figura 3.1: Combinaciones *top*, *min* y *bottom* 53

Universitat d'Alacant
Universidad de Alicante

Tabla de acrónimos

0r ..... *zero replacement*
AES ..... advanced encryption standard
ARC4 ..... *alleged* Rivest *cipher* 4
ARX ..... *add-rotate-xor*
AVX ..... a*dvanced vector extensions*
BBS ..... Blum Blum Shub
bms ..... bit menos significativo
bpi ..... bits por iteración
CBC ..... *cipher block chaining* (modo)
CFB ..... *cipher feedback* (modo)
CTR ..... *counter* (modo)
ECB ..... *electronic codebook* (modo)
ECRYPT ..... *European Network of Excellence for Cryptology*
eStream ..... *the ECRYPT Stream Cipher Project*
FPGA ..... f*ield programmable gate array*
GPGPU ..... g*eneral purpose computing on graphics processing units*
GPU ..... g*raphics processing unit*
HC128 ..... Hongjun*'s Cipher* 128
HC256 ..... Hongjun*'s Cipher* 256
KSA ..... k*ey scheduling algorithm*
LCG ..... *linear congruential generator*
LFSR ..... *linear feedback shift register*
mbp ..... mínimo de bits para el periodo
mbs ..... mínimo de bit para la semilla
min ..... combinación con *tbn* mínimo y $mbs \geq 128$
NLFSR ..... *non-linear feedback shift register*
OFB ..... *output feedback* (modo)
OTP ..... *one-time pad*
PCBC ..... *propagating cipher block chaining* (modo)
PIPS ..... *Primitive and Irreducible Polynomial Server*
PNI ..... Prescott *new instructions*
PractRand ..... *Practically Random*
PRGA ..... *pseudo-random generation algorithm*
Primpoly ..... A Program for Computing Primitive Polynomials
RC4 ..... Rivest *cipher* 4
RFC ..... request for comments
s-box ..... caja de sustitución
SIMD ..... *single instruction, multiple data*
SSE ..... *streaming* SIMD *extensions*
SSL ..... *secure sockets layer*
tbn ..... tamaño de los bloques no nulos
TSB ..... triangular superior por bloques (matriz)
TSL ..... *transport layer security*
WEP ..... *wired equivalent privacy*

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Apéndice A
Baterías de experimentación

A.1 Primera batería

En una primera tabla se muestran las combinaciones *top*, *min* y *bottom* (véanse definición 3.13, definición 3.14 y definición 3.15) elegidas para realizar la primera batería, indicando el mínimo de bits para el periodo (*mbp*), para la semilla (*mbs*) y el tamaño de los bloques no nulos (*tbn*).

Las siguientes tablas muestran los resultados de los distintos test aplicados a las secuencias obtenidas en las pruebas. Las columnas *slow-n*, *slow-f*, *fast-n* y *fast-f* se refieren a los sistemas de extracción *slow* y *fast* sin (*-n*) y con (*-f*) filtrado.

Universitat d'Alacant
Universidad de Alicante

Para cada test se establecen dos valores de corrección que no se deben superar, según el nivel de significación (definición 1.34) establecido, para no ser rechazado. Los dos niveles de significación son $\alpha = 0{,}1$, el más restrictivo que supondría rechazar hasta un 10% de falsos positivos, y $\alpha = 0{,}001$, en el que sólo en un 0,1% se rechazaría una secuencia como no aleatoria de forma equivocada.

Se compara además con valores estadísticos de referencia para BBS, AES y RC4 (obtenidos de [5]). Se indica con el color de la celda si la secuencia es rechazada como aleatoria, en anaranjado para el nivel de significación $\alpha = 0{,}001$, y en amarillo para $\alpha = 0{,}1$. Los resultados que superan o igualan al mejor de los tres algoritmos comparados, se muestran con texto en verde. Los test efectuados en esta batería de prueba son el de frecuencia (o monobit), serial (o de pareja), póker (con 8 y 16), rachas y autocorrelación. Además se pasa también un test no estadístico, el de la complejidad lineal, cuyo resultado debe coincidir con la mitad de la secuencia. Véase la sección 1.3 para más información.

Una última tabla muestra el coste, en mínimo de ciclos de CPU (de las 10 semillas usadas), para todas las combinaciones probadas, extracción *slow* o *fast* con o sin filtrado.

La información de los resultados se encuentra disponible para su tratamiento en el enlace:

http://www.dccia.ua.es/~fmartine/tesis/Bateria1.zip

Universitat d'Alacant
Universidad de Alicante

Combinaciones probadas en la primera batería

top

p	*s*	*r*	*t*	*mbp*	*mbs*	*tbn*
3	5	6	7	26	169	217
5	5	6	7	37	248	217
7	5	6	7	45	300	217
11	5	6	7	55	370	217
13	5	6	7	59	395	217
17	5	6	7	65	437	217
19	5	6	7	68	454	217
23	5	6	7	72	484	217
29	5	6	7	77	519	217
31	5	6	7	79	530	217
37	5	6	7	83	557	217
41	5	6	7	85	573	217
43	5	6	7	86	580	217
47	5	6	7	88	594	217
53	5	6	7	91	612	217
59	5	6	7	94	629	217
61	5	6	7	94	634	217
67	5	6	7	97	649	217
71	5	6	7	98	658	217
73	5	6	7	99	662	217
79	5	6	7	100	674	217
83	5	6	7	102	682	217
89	5	6	7	103	692	217
97	5	6	7	105	706	217
101	5	6	7	106	712	217
103	5	6	7	107	715	217
107	5	6	7	107	721	217
109	5	6	7	108	724	217
113	5	6	7	109	729	217
127	5	6	7	111	747	217
131	5	6	7	112	752	217
137	5	6	7	113	759	217
139	5	6	7	113	761	217
149	5	6	7	115	772	217
151	5	6	7	115	774	217
157	5	6	7	116	780	217
163	5	6	7	117	786	217
167	5	6	7	118	790	217
173	4	5	6	84	550	151
179	5	6	7	119	800	217
181	5	6	7	120	802	217
191	5	6	7	121	810	217
193	5	6	7	121	812	217
197	5	6	7	121	815	217
199	5	6	7	122	817	217
211	5	6	7	123	826	217
223	5	6	7	124	834	217
227	5	6	7	125	837	217
229	5	6	7	125	838	217
233	5	6	7	125	841	217
239	5	6	7	126	845	217
241	5	6	7	126	846	217
251	5	6	7	127	852	217
257	4	5	7	108	664	173
263	5	6	7	128	860	217
269	5	6	7	129	863	217
271	4	5	6	95	598	151
277	5	6	7	129	868	217
281	5	6	7	130	870	217
283	5	6	7	130	871	217
293	4	5	6	90	606	151
307	5	6	7	132	884	217
311	5	6	7	132	886	217
313	5	6	7	132	887	217

min

p	*s*	*r*	*t*	*mbp*	*mbs*	*tbn*
3	4	5	7	23	131	173
5	3	5	6	23	146	133
7	3	4	5	28	131	97
11	2	4	5	27	131	83
13	2	4	5	29	140	83
17	2	4	5	32	155	83
19	2	4	5	32	161	83
23	2	3	5	36	140	69
29	2	3	5	38	150	69
31	2	3	4	29	128	55
37	2	3	4	29	135	55
41	2	3	4	32	139	55
43	2	3	4	32	141	55
47	2	3	4	29	144	55
53	2	3	4	34	148	55
59	2	3	4	35	152	55
61	2	3	4	35	154	55
67	2	3	4	36	157	55
71	2	3	4	34	159	55
73	2	3	4	37	160	55
79	2	3	4	35	163	55
83	2	3	4	38	165	55
89	2	3	4	38	168	55
97	2	3	4	39	171	55
101	2	3	4	36	173	55
103	2	3	4	40	173	55
107	1	3	4	40	128	45
109	1	3	4	40	128	45
113	1	3	4	40	129	45
127	1	3	4	41	132	45
131	1	3	4	42	133	45
137	1	3	4	42	134	45
139	1	3	4	42	135	45
149	1	3	4	43	137	45
151	1	3	4	43	137	45
157	1	3	4	43	138	45
163	1	3	4	44	139	45
167	1	3	4	44	140	45
173	1	3	4	44	141	45
179	1	3	4	44	142	45
181	1	3	4	45	142	45
191	1	3	4	45	143	45
193	1	3	4	45	144	45
197	1	3	4	45	144	45
199	1	3	4	45	145	45
211	1	3	4	46	146	45
223	1	3	4	46	148	45
227	1	3	4	46	148	45
229	1	3	4	47	148	45
233	1	3	4	47	149	45
239	1	3	4	47	150	45
241	1	3	4	47	150	45
251	1	3	4	47	151	45
257	1	3	4	48	152	45
263	1	3	4	48	152	45
269	1	3	4	48	153	45
271	1	3	4	48	153	45
277	1	3	4	48	154	45
281	1	3	4	48	154	45
283	1	3	4	48	154	45
293	1	3	4	49	155	45
307	1	3	4	49	156	45
311	1	3	4	49	157	45
313	1	3	4	49	157	45

bottom

p	*s*	*r*	*t*	*mbp*	*mbs*	*tbn*
3	4	5	7	23	131	173
5	2	5	7	28	136	137
7	1	5	7	31	131	122
11	1	4	7	34	134	105
13	1	5	6	37	151	103
17	1	4	6	32	138	87
19	1	3	7	38	131	90
23	1	3	7	40	140	90
29	1	3	7	43	150	90
31	1	3	7	44	153	90
37	1	3	7	46	161	90
41	1	3	7	48	166	90
43	1	3	7	48	168	90
47	1	3	7	50	172	90
53	1	2	7	45	131	77
59	1	3	5	41	135	58
61	1	2	7	47	136	77
67	1	3	5	42	139	58
71	1	3	5	43	141	58
73	1	2	7	49	142	77
79	1	3	5	44	144	58
83	1	3	5	44	146	58
89	1	3	5	45	148	58
97	1	2	7	52	151	77
101	1	3	5	46	153	58
103	1	2	7	53	153	77
107	1	3	4	40	128	45
109	1	3	4	40	128	45
113	1	3	4	40	129	45
127	1	3	4	41	132	45
131	1	3	4	42	133	45
137	1	2	7	56	163	77
139	1	3	4	42	135	45
149	1	3	4	43	137	45
151	1	2	7	57	166	77
157	1	2	7	58	167	77
163	1	2	7	58	169	77
167	1	3	4	44	140	45
173	1	3	4	42	141	45
179	1	3	4	44	142	45
181	1	3	4	45	142	45
191	1	3	4	45	143	45
193	1	2	5	45	129	47
197	1	3	4	45	144	45
199	1	3	4	45	145	45
211	1	2	5	46	131	47
223	1	3	4	46	148	45
227	1	3	4	46	148	45
229	1	2	5	47	133	47
233	1	3	4	47	149	45
239	1	3	4	47	150	45
241	1	2	5	47	134	47
251	1	3	4	47	151	45
257	1	2	5	48	136	47
263	1	2	5	48	136	47
269	1	3	4	48	153	45
271	1	2	5	48	137	47
277	1	2	5	48	137	47
281	1	2	5	48	138	47
283	1	2	5	48	138	47
293	1	2	5	49	139	47
307	1	3	4	49	156	45
311	1	3	4	49	157	45
313	1	2	5	49	140	47

Tabla del test de frecuencia para combinaciones *top*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r5s6t7	1,11052	1,03734	2180,82952	1,2865	1,0646	1,0268	1,1035	2,706	10,83
5	p5r5s6t7	1,96348	1,35038	789,16642	1,07464	1,0646	1,0268	1,1035	2,706	10,83
7	p7r5s6t7	1,0118	0,98048	410,27256	2,39332	1,0646	1,0268	1,1035	2,706	10,83
11	p11r5s6t7	0,90036	1,24286	174,77186	1,3024	1,0646	1,0268	1,1035	2,706	10,83
13	p13r5s6t7	0,62292	1,37446	131,52922	2,09076	1,0646	1,0268	1,1035	2,706	10,83
17	p17r5s6t7	0,75082	0,88154	66,79712	1,31512	1,0646	1,0268	1,1035	2,706	10,83
19	p19r5s6t7	0,63192	0,8139	64,3483	1,47352	1,0646	1,0268	1,1035	2,706	10,83
23	p23r5s6t7	0,97584	0,82632	38,3309	1,29372	1,0646	1,0268	1,1035	2,706	10,83
29	p29r5s6t7	0,76892	0,97756	28,18796	0,20736	1,0646	1,0268	1,1035	2,706	10,83
31	p31r5s6t7	0,45248	1,00526	27,80102	1,30818	1,0646	1,0268	1,1035	2,706	10,83
37	p37r5s6t7	1,82766	0,98728	13,66374	1,19062	1,0646	1,0268	1,1035	2,706	10,83
41	p41r5s6t7	0,83456	0,3468	9,53626	0,96392	1,0646	1,0268	1,1035	2,706	10,83
43	p43r5s6t7	0,7862	1,444	15,68166	1,26818	1,0646	1,0268	1,1035	2,706	10,83
47	p47r5s6t7	1,16306	1,33334	9,91718	0,6397	1,0646	1,0268	1,1035	2,706	10,83
53	p53r5s6t7	1,56164	1,3452	8,91694	0,99908	1,0646	1,0268	1,1035	2,706	10,83
59	p59r5s6t7	1,03034	0,5326	6,213	0,74156	1,0646	1,0268	1,1035	2,706	10,83
61	p61r5s6t7	0,43684	1,47254	4,2395	1,28808	1,0646	1,0268	1,1035	2,706	10,83
67	p67r5s6t7	0,92046	0,60732	5,88302	0,97152	1,0646	1,0268	1,1035	2,706	10,83
71	p71r5s6t7	1,32758	0,71676	3,58406	0,55908	1,0646	1,0268	1,1035	2,706	10,83
73	p73r5s6t7	0,97762	0,77854	3,60658	1,2895	1,0646	1,0268	1,1035	2,706	10,83
79	p79r5s6t7	0,82056	0,91454	6,59758	0,74094	1,0646	1,0268	1,1035	2,706	10,83
83	p83r5s6t7	1,23186	0,82734	2,8145	0,89868	1,0646	1,0268	1,1035	2,706	10,83
89	p89r5s6t7	0,71828	0,89958	1,74254	2,4414	1,0646	1,0268	1,1035	2,706	10,83
97	p97r5s6t7	1,09572	1,12762	1,61058	0,72428	1,0646	1,0268	1,1035	2,706	10,83
101	p101r5s6t7	0,67586	1,52806	2,58722	0,61552	1,0646	1,0268	1,1035	2,706	10,83
103	p103r5s6t7	0,8545	0,81392	2,65904	0,67264	1,0646	1,0268	1,1035	2,706	10,83
107	p107r5s6t7	0,4329	1,20288	2,31022	0,45802	1,0646	1,0268	1,1035	2,706	10,83
109	p109r5s6t7	0,88188	0,78762	2,8415	0,6491	1,0646	1,0268	1,1035	2,706	10,83
113	p113r5s6t7	1,27774	1,52468	2,34156	0,901	1,0646	1,0268	1,1035	2,706	10,83
127	p127r5s6t7	0,79438	1,03602	1,86778	0,386	1,0646	1,0268	1,1035	2,706	10,83
131	p131r5s6t7	0,8199	0,92594	2,57868	0,9668	1,0646	1,0268	1,1035	2,706	10,83
137	p137r5s6t7	0,25134	1,91608	3,04876	0,67214	1,0646	1,0268	1,1035	2,706	10,83
139	p139r5s6t7	0,81468	0,63762	1,2179	1,17998	1,0646	1,0268	1,1035	2,706	10,83
149	p149r5s6t7	1,11536	1,55302	0,69902	0,57764	1,0646	1,0268	1,1035	2,706	10,83
151	p151r5s6t7	1,44344	0,51582	1,68108	0,31542	1,0646	1,0268	1,1035	2,706	10,83
157	p157r5s6t7	2,00408	0,41978	2,84754	0,77634	1,0646	1,0268	1,1035	2,706	10,83
163	p163r5s6t7	0,93486	0,92786	1,9814	0,6663	1,0646	1,0268	1,1035	2,706	10,83
167	p167r5s6t7	1,45296	1,2877	3,12458	1,37474	1,0646	1,0268	1,1035	2,706	10,83
173	p173r4s5t6	0,89834	0,78048	2,70482	0,58938	1,0646	1,0268	1,1035	2,706	10,83
179	p179r5s6t7	2,27892	0,51024	1,5784	1,4112	1,0646	1,0268	1,1035	2,706	10,83
181	p181r5s6t7	0,85576	1,31392	1,58856	0,9501	1,0646	1,0268	1,1035	2,706	10,83
191	p191r5s6t7	0,765	0,69852	2,22368	1,86048	1,0646	1,0268	1,1035	2,706	10,83
193	p193r5s6t7	0,81262	0,24292	1,57636	0,28936	1,0646	1,0268	1,1035	2,706	10,83
197	p197r5s6t7	1,63902	0,59966	1,20782	0,61704	1,0646	1,0268	1,1035	2,706	10,83
199	p199r5s6t7	0,68258	1,32414	2,5966	0,49706	1,0646	1,0268	1,1035	2,706	10,83
211	p211r5s6t7	1,85952	1,1014	2,51342	0,6604	1,0646	1,0268	1,1035	2,706	10,83
223	p223r5s6t7	1,12704	0,94418	1,73482	1,58752	1,0646	1,0268	1,1035	2,706	10,83
227	p227r5s6t7	0,76916	0,99912	1,17906	0,48116	1,0646	1,0268	1,1035	2,706	10,83
229	p229r5s6t7	0,49398	0,38578	1,12744	1,01802	1,0646	1,0268	1,1035	2,706	10,83
233	p233r5s6t7	0,79308	1,42892	0,72644	1,19776	1,0646	1,0268	1,1035	2,706	10,83
239	p239r5s6t7	0,40692	1,88236	1,13614	0,70608	1,0646	1,0268	1,1035	2,706	10,83
241	p241r5s6t7	1,81604	0,41622	3,0257	1,68126	1,0646	1,0268	1,1035	2,706	10,83
251	p251r5s6t7	0,67668	0,61484	0,79962	0,63314	1,0646	1,0268	1,1035	2,706	10,83
257	p257r4s5t7	1,44694	0,60366	1,27046	0,65728	1,0646	1,0268	1,1035	2,706	10,83
263	p263r5s6t7	1,3184	1,3723	1,98222	0,8959	1,0646	1,0268	1,1035	2,706	10,83
269	p269r5s6t7	0,52158	1,4736	1,33398	0,99776	1,0646	1,0268	1,1035	2,706	10,83
271	p271r4s5t6	1,22254	0,7957	0,67056	1,0325	1,0646	1,0268	1,1035	2,706	10,83
277	p277r5s6t7	0,70494	0,68848	1,17168	2,76846	1,0646	1,0268	1,1035	2,706	10,83
281	p281r5s6t7	1,6006	0,68516	1,05026	0,87796	1,0646	1,0268	1,1035	2,706	10,83
283	p283r5s6t7	0,72974	1,9624	0,79358	0,64524	1,0646	1,0268	1,1035	2,706	10,83
293	p293r4s6t7	1,022	1,23596	1,15034	1,75762	1,0646	1,0268	1,1035	2,706	10,83
307	p307r5s6t7	0,53802	1,18138	0,72982	0,5753	1,0646	1,0268	1,1035	2,706	10,83
311	p311r5s6t7	1,51826	0,50762	0,61126	0,33824	1,0646	1,0268	1,1035	2,706	10,83
313	p313r5s6t7	0,92482	1,53614	0,98852	0,95406	1,0646	1,0268	1,1035	2,706	10,83

Tabla del test de frecuencia para combinaciones *min*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	0,74002	0,92834	2170,32118	1,2362	1,0646	1,0268	1,1035	2,706	10,83
5	p5r3s5t6	1,02708	0,82128	807,6215	0,93688	1,0646	1,0268	1,1035	2,706	10,83
7	p7r3s4t5	1,0031	1,23358	421,24808	2,0082	1,0646	1,0268	1,1035	2,706	10,83
11	p11r2s4t5	0,47074	0,6125	159,6546	2,1798	1,0646	1,0268	1,1035	2,706	10,83
13	p13r2s4t5	0,72804	0,96084	125,79198	0,89008	1,0646	1,0268	1,1035	2,706	10,83
17	p17r2s4t5	2,46884	0,59022	69,52908	1,52648	1,0646	1,0268	1,1035	2,706	10,83
19	p19r2s4t5	0,67038	0,90816	50,10122	0,65404	1,0646	1,0268	1,1035	2,706	10,83
23	p23r2s3t5	1,36562	0,9482	41,80394	0,63868	1,0646	1,0268	1,1035	2,706	10,83
29	p29r2s3t5	1,46852	1,49786	21,51678	1,69804	1,0646	1,0268	1,1035	2,706	10,83
31	p31r2s3t4	0,48634	0,61398	24,71922	0,97442	1,0646	1,0268	1,1035	2,706	10,83
37	p37r2s3t4	0,93464	0,67152	16,54042	1,262	1,0646	1,0268	1,1035	2,706	10,83
41	p41r2s3t4	0,32818	0,6954	12,5012	0,6745	1,0646	1,0268	1,1035	2,706	10,83
43	p43r2s3t4	0,5819	1,02892	10,31956	1,4609	1,0646	1,0268	1,1035	2,706	10,83
47	p47r2s3t4	1,49202	0,61438	7,19086	0,80626	1,0646	1,0268	1,1035	2,706	10,83
53	p53r2s3t4	0,53472	0,33866	6,2808	1,36322	1,0646	1,0268	1,1035	2,706	10,83
59	p59r2s3t4	0,9027	0,81622	8,49552	0,70794	1,0646	1,0268	1,1035	2,706	10,83
61	p61r2s3t4	0,98578	0,36804	5,01608	0,86004	1,0646	1,0268	1,1035	2,706	10,83
67	p67r2s3t4	1,57628	1,79242	5,01892	1,31994	1,0646	1,0268	1,1035	2,706	10,83
71	p71r2s3t4	1,55302	1,75164	4,82916	0,84908	1,0646	1,0268	1,1035	2,706	10,83
73	p73r2s3t4	0,73452	0,97444	3,99886	0,9931	1,0646	1,0268	1,1035	2,706	10,83
79	p79r2s3t4	1,06818	0,75996	4,38746	1,02008	1,0646	1,0268	1,1035	2,706	10,83
83	p83r2s3t4	0,57126	1,63742	5,15626	1,20044	1,0646	1,0268	1,1035	2,706	10,83
89	p89r2s3t4	0,5496	1,27614	0,56296	0,49734	1,0646	1,0268	1,1035	2,706	10,83
97	p97r2s3t4	1,19672	1,03478	4,56534	0,4173	1,0646	1,0268	1,1035	2,706	10,83
101	p101r2s3t4	1,26298	1,19582	2,78314	0,82356	1,0646	1,0268	1,1035	2,706	10,83
103	p103r2s3t4	0,85956	1,3925	2,65732	1,56736	1,0646	1,0268	1,1035	2,706	10,83
107	p107r1s3t4	1,15486	0,92678	3,03848	0,3866	1,0646	1,0268	1,1035	2,706	10,83
109	p109r1s3t4	1,41786	0,72232	3,04194	1,19174	1,0646	1,0268	1,1035	2,706	10,83
113	p113r1s3t4	0,76196	1,11602	2,26724	1,73852	1,0646	1,0268	1,1035	2,706	10,83
127	p127r1s3t4	1,2373	0,51596	2,835	0,49556	1,0646	1,0268	1,1035	2,706	10,83
131	p131r1s3t4	1,20964	1,21992	2,5952	0,95344	1,0646	1,0268	1,1035	2,706	10,83
137	p137r1s3t4	0,697	1,08052	1,31844	1,23658	1,0646	1,0268	1,1035	2,706	10,83
139	p139r1s3t4	0,42106	1,1522	1,55178	1,121	1,0646	1,0268	1,1035	2,706	10,83
149	p149r1s3t4	2,26096	1,11678	1,7066	1,59272	1,0646	1,0268	1,1035	2,706	10,83
151	p151r1s3t4	0,82216	0,5349	3,10904	1,2668	1,0646	1,0268	1,1035	2,706	10,83
157	p157r1s3t4	1,2092	0,36014	0,91606	0,97506	1,0646	1,0268	1,1035	2,706	10,83
163	p163r1s3t4	0,79226	0,54486	1,50896	0,6911	1,0646	1,0268	1,1035	2,706	10,83
167	p167r1s3t4	0,59298	0,84892	1,13764	0,23818	1,0646	1,0268	1,1035	2,706	10,83
173	p173r1s3t4	0,2871	1,43332	1,15138	1,96032	1,0646	1,0268	1,1035	2,706	10,83
179	p179r1s3t4	1,21188	1,59914	1,6363	1,98908	1,0646	1,0268	1,1035	2,706	10,83
181	p181r1s3t4	0,78526	0,407	1,93814	0,5385	1,0646	1,0268	1,1035	2,706	10,83
191	p191r1s3t4	0,41006	0,5864	1,06438	1,02132	1,0646	1,0268	1,1035	2,706	10,83
193	p193r1s3t4	1,22812	0,95474	1,35652	0,7154	1,0646	1,0268	1,1035	2,706	10,83
197	p197r1s3t4	1,35176	0,68446	2,38376	0,66492	1,0646	1,0268	1,1035	2,706	10,83
199	p199r1s3t4	1,25066	1,2054	1,44548	1,31848	1,0646	1,0268	1,1035	2,706	10,83
211	p211r1s3t4	1,43344	1,0205	1,7081	2,09332	1,0646	1,0268	1,1035	2,706	10,83
223	p223r1s3t4	0,27778	0,6917	0,45742	0,4574	1,0646	1,0268	1,1035	2,706	10,83
227	p227r1s3t4	0,8405	1,35804	1,84122	0,24044	1,0646	1,0268	1,1035	2,706	10,83
229	p229r1s3t4	1,39242	0,68496	1,07318	0,50582	1,0646	1,0268	1,1035	2,706	10,83
233	p233r1s3t4	1,1187	0,88804	1,59964	0,72608	1,0646	1,0268	1,1035	2,706	10,83
239	p239r1s3t4	1,06838	1,0551	1,89748	1,04948	1,0646	1,0268	1,1035	2,706	10,83
241	p241r1s3t4	1,40404	1,1824	0,59452	1,14976	1,0646	1,0268	1,1035	2,706	10,83
251	p251r1s3t4	0,448	0,50656	2,2969	0,67146	1,0646	1,0268	1,1035	2,706	10,83
257	p257r1s3t4	0,27576	0,93602	1,35898	1,33688	1,0646	1,0268	1,1035	2,706	10,83
263	p263r1s3t4	0,85594	1,38918	1,54018	0,54536	1,0646	1,0268	1,1035	2,706	10,83
269	p269r1s3t4	0,98234	1,3129	0,47206	0,75216	1,0646	1,0268	1,1035	2,706	10,83
271	p271r1s3t4	1,1661	0,62142	1,53188	1,28786	1,0646	1,0268	1,1035	2,706	10,83
277	p277r1s3t4	1,63616	1,20096	0,74432	1,1752	1,0646	1,0268	1,1035	2,706	10,83
281	p281r1s3t4	1,02766	1,111	1,48766	1,3832	1,0646	1,0268	1,1035	2,706	10,83
283	p283r1s3t4	0,47892	0,7855	1,09176	1,78266	1,0646	1,0268	1,1035	2,706	10,83
293	p293r1s3t4	1,47912	0,9593	1,54492	1,2438	1,0646	1,0268	1,1035	2,706	10,83
307	p307r1s3t4	0,55698	0,76192	1,47584	0,50512	1,0646	1,0268	1,1035	2,706	10,83
311	p311r1s3t4	1,26246	1,12142	1,1517	1,44408	1,0646	1,0268	1,1035	2,706	10,83
313	p313r1s3t4	0,86788	1,38166	2,21092	1,79646	1,0646	1,0268	1,1035	2,706	10,83

Tabla del test de frecuencia para combinaciones *bottom*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	0,74002	0,92834	2170,32118	1,2362	1,0646	1,0268	1,1035	2,706	10,83
5	p5r2s5t7	0,53898	1,40668	798,15276	1,41698	1,0646	1,0268	1,1035	2,706	10,83
7	p7r1s5t7	0,98044	1,55634	427,66192	2,08516	1,0646	1,0268	1,1035	2,706	10,83
11	p11r1s4t7	0,3606	1,79952	162,2536	0,94456	1,0646	1,0268	1,1035	2,706	10,83
13	p13r1s5t6	0,92206	0,62648	103,975	2,12598	1,0646	1,0268	1,1035	2,706	10,83
17	p17r1s4t6	0,5194	0,87832	71,24552	0,99408	1,0646	1,0268	1,1035	2,706	10,83
19	p19r1s3t7	0,60448	0,24546	54,0621	1,2212	1,0646	1,0268	1,1035	2,706	10,83
23	p23r1s3t7	1,04762	0,74724	41,22518	1,2564	1,0646	1,0268	1,1035	2,706	10,83
29	p29r1s3t7	1,7786	1,13452	22,6247	0,4702	1,0646	1,0268	1,1035	2,706	10,83
31	p31r1s3t7	0,73208	1,21502	18,55336	1,50454	1,0646	1,0268	1,1035	2,706	10,83
37	p37r1s3t7	0,4889	0,86748	17,25824	0,8803	1,0646	1,0268	1,1035	2,706	10,83
41	p41r1s3t7	0,7615	0,7752	10,59604	0,68674	1,0646	1,0268	1,1035	2,706	10,83
43	p43r1s3t7	0,62344	0,3962	20,37124	1,03102	1,0646	1,0268	1,1035	2,706	10,83
47	p47r1s3t7	0,53338	0,9451	11,5256	0,9956	1,0646	1,0268	1,1035	2,706	10,83
53	p53r1s2t7	1,09044	1,02836	6,22496	0,85146	1,0646	1,0268	1,1035	2,706	10,83
59	p59r1s3t5	1,5674	1,09652	6,30768	0,8344	1,0646	1,0268	1,1035	2,706	10,83
61	p61r1s2t7	0,50648	1,48024	8,76948	2,51782	1,0646	1,0268	1,1035	2,706	10,83
67	p67r1s3t5	1,02374	0,87224	5,14994	0,78334	1,0646	1,0268	1,1035	2,706	10,83
71	p71r1s3t5	0,78878	1,34222	1,76138	1,48422	1,0646	1,0268	1,1035	2,706	10,83
73	p73r1s2t7	0,794	0,66332	4,68746	0,58386	1,0646	1,0268	1,1035	2,706	10,83
79	p79r1s3t5	1,4871	2,14728	4,50046	0,53822	1,0646	1,0268	1,1035	2,706	10,83
83	p83r1s3t5	1,20104	0,73724	3,71634	0,67206	1,0646	1,0268	1,1035	2,706	10,83
89	p89r1s3t5	0,8345	0,82106	4,62508	0,62544	1,0646	1,0268	1,1035	2,706	10,83
97	p97r1s2t7	1,27838	0,54722	0,70398	0,91362	1,0646	1,0268	1,1035	2,706	10,83
101	p101r1s3t5	0,7633	0,77738	1,81242	2,45696	1,0646	1,0268	1,1035	2,706	10,83
103	p103r1s2t7	0,98788	1,15972	1,87536	1,03002	1,0646	1,0268	1,1035	2,706	10,83
107	p107r1s3t4	1,15486	0,92678	3,03848	0,3866	1,0646	1,0268	1,1035	2,706	10,83
109	p109r1s3t4	1,41786	0,72232	3,04194	1,19174	1,0646	1,0268	1,1035	2,706	10,83
113	p113r1s3t4	0,76196	1,11602	2,26724	1,73852	1,0646	1,0268	1,1035	2,706	10,83
127	p127r1s3t4	1,2373	0,51596	2,835	0,49556	1,0646	1,0268	1,1035	2,706	10,83
131	p131r1s3t4	1,20964	1,21992	2,5952	0,95344	1,0646	1,0268	1,1035	2,706	10,83
137	p137r1s2t7	0,70952	1,1059	1,61042	0,79666	1,0646	1,0268	1,1035	2,706	10,83
139	p139r1s3t4	0,42106	1,1522	1,55178	1,121	1,0646	1,0268	1,1035	2,706	10,83
149	p149r1s3t4	2,26096	1,11678	1,7066	1,59272	1,0646	1,0268	1,1035	2,706	10,83
151	p151r1s2t7	0,60612	0,55462	3,37456	0,97004	1,0646	1,0268	1,1035	2,706	10,83
157	p157r1s2t7	0,62268	0,43212	3,27436	0,95702	1,0646	1,0268	1,1035	2,706	10,83
163	p163r1s2t7	0,79656	1,81574	1,60646	0,60376	1,0646	1,0268	1,1035	2,706	10,83
167	p167r1s3t4	0,59298	0,84892	1,13764	0,23818	1,0646	1,0268	1,1035	2,706	10,83
173	p173r1s3t4	0,2871	1,43332	1,15138	1,96032	1,0646	1,0268	1,1035	2,706	10,83
179	p179r1s3t4	1,21188	1,59914	1,6363	1,98908	1,0646	1,0268	1,1035	2,706	10,83
181	p181r1s3t4	0,78526	0,407	1,93814	0,5385	1,0646	1,0268	1,1035	2,706	10,83
191	p191r1s3t4	0,41006	0,5864	1,06438	1,02132	1,0646	1,0268	1,1035	2,706	10,83
193	p193r1s2t5	0,77018	1,43508	1,89484	0,8077	1,0646	1,0268	1,1035	2,706	10,83
197	p197r1s3t4	1,35176	0,68446	2,38376	0,66492	1,0646	1,0268	1,1035	2,706	10,83
199	p199r1s3t4	1,25066	1,2054	1,44548	1,31848	1,0646	1,0268	1,1035	2,706	10,83
211	p211r1s2t5	1,49822	0,5513	3,06452	0,9985	1,0646	1,0268	1,1035	2,706	10,83
223	p223r1s3t4	0,27778	0,6917	0,45742	0,4574	1,0646	1,0268	1,1035	2,706	10,83
227	p227r1s3t4	0,8405	1,35804	1,84122	0,24044	1,0646	1,0268	1,1035	2,706	10,83
229	p229r1s2t5	1,59364	0,62192	0,79572	0,75776	1,0646	1,0268	1,1035	2,706	10,83
233	p233r1s3t4	1,1187	0,88804	1,59964	0,72608	1,0646	1,0268	1,1035	2,706	10,83
239	p239r1s3t4	1,06838	1,0551	1,89748	1,04948	1,0646	1,0268	1,1035	2,706	10,83
241	p241r1s2t5	1,35956	1,8704	0,86768	1,31006	1,0646	1,0268	1,1035	2,706	10,83
251	p251r1s3t4	0,448	0,50656	2,2969	0,67146	1,0646	1,0268	1,1035	2,706	10,83
257	p257r1s2t5	1,33092	0,25168	1,4289	0,64728	1,0646	1,0268	1,1035	2,706	10,83
263	p263r1s2t5	1,46882	0,81162	1,37128	1,79354	1,0646	1,0268	1,1035	2,706	10,83
269	p269r1s3t4	0,98234	1,3129	0,47206	0,75216	1,0646	1,0268	1,1035	2,706	10,83
271	p271r1s2t5	1,19136	0,93996	1,49124	1,40248	1,0646	1,0268	1,1035	2,706	10,83
277	p277r1s2t5	1,63512	1,03724	1,49594	1,00558	1,0646	1,0268	1,1035	2,706	10,83
281	p281r1s2t5	0,41024	0,8347	0,9727	0,41082	1,0646	1,0268	1,1035	2,706	10,83
283	p283r1s2t5	0,63944	0,67928	0,74482	1,14158	1,0646	1,0268	1,1035	2,706	10,83
293	p293r1s2t5	1,31734	1,03934	1,22186	1,12216	1,0646	1,0268	1,1035	2,706	10,83
307	p307r1s3t4	0,55698	0,76192	1,47584	0,50512	1,0646	1,0268	1,1035	2,706	10,83
311	p311r1s3t4	1,26246	1,12142	1,1517	1,44408	1,0646	1,0268	1,1035	2,706	10,83
313	p313r1s2t5	0,84096	0,82262	2,24236	0,58	1,0646	1,0268	1,1035	2,706	10,83

Tabla del test serial para combinaciones *top*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r5s6t7	2,29808	2,14314	2409,42754	2181,97595	1,5252	1,4981	2,2746	4,605	13,82
5	p5r5s6t7	2,78518	3,30472	819,1288	790,04067	1,5252	1,4981	2,2746	4,605	13,82
7	p7r5s6t7	1,94803	1,98382	418,10453	412,52853	1,5252	1,4981	2,2746	4,605	13,82
11	p11r5s6t7	1,46599	2,15092	177,25566	176,03722	1,5252	1,4981	2,2746	4,605	13,82
13	p13r5s6t7	1,00788	1,99601	132,7475	133,4862	1,5252	1,4981	2,2746	4,605	13,82
17	p17r5s6t7	1,8955	1,63443	67,88778	68,08218	1,5252	1,4981	2,2746	4,605	13,82
19	p19r5s6t7	1,25745	1,44352	65,84981	65,85139	1,5252	1,4981	2,2746	4,605	13,82
23	p23r5s6t7	1,62729	1,81185	39,16713	39,64563	1,5252	1,4981	2,2746	4,605	13,82
29	p29r5s6t7	1,2445	1,74191	29,39703	28,41357	1,5252	1,4981	2,2746	4,605	13,82
31	p31r5s6t7	1,04383	1,45286	29,0515	29,13972	1,5252	1,4981	2,2746	4,605	13,82
37	p37r5s6t7	2,43448	2,80746	14,71753	14,81298	1,5252	1,4981	2,2746	4,605	13,82
41	p41r5s6t7	1,67941	1,17455	11,19222	10,48609	1,5252	1,4981	2,2746	4,605	13,82
43	p43r5s6t7	2,46809	2,2299	16,38573	16,94488	1,5252	1,4981	2,2746	4,605	13,82
47	p47r5s6t7	1,59944	2,50264	10,88536	10,55495	1,5252	1,4981	2,2746	4,605	13,82
53	p53r5s6t7	2,23382	2,89038	9,58224	9,93513	1,5252	1,4981	2,2746	4,605	13,82
59	p59r5s6t7	1,86176	1,56638	6,9708	6,9465	1,5252	1,4981	2,2746	4,605	13,82
61	p61r5s6t7	1,28027	1,89211	5,29241	5,51685	1,5252	1,4981	2,2746	4,605	13,82
67	p67r5s6t7	1,49827	1,52326	6,67752	6,83739	1,5252	1,4981	2,2746	4,605	13,82
71	p71r5s6t7	2,09162	2,05018	4,5377	4,13503	1,5252	1,4981	2,2746	4,605	13,82
73	p73r5s6t7	2,01165	1,75745	4,54058	4,89804	1,5252	1,4981	2,2746	4,605	13,82
79	p79r5s6t7	2,64608	1,74171	7,11238	7,33272	1,5252	1,4981	2,2746	4,605	13,82
83	p83r5s6t7	2,52041	2,04575	4,10299	3,69387	1,5252	1,4981	2,2746	4,605	13,82
89	p89r5s6t7	2,11614	1,61251	2,51178	4,1862	1,5252	1,4981	2,2746	4,605	13,82
97	p97r5s6t7	1,48376	2,22515	2,0066	2,33186	1,5252	1,4981	2,2746	4,605	13,82
101	p101r5s6t7	2,02092	2,18952	3,17157	3,2025	1,5252	1,4981	2,2746	4,605	13,82
103	p103r5s6t7	1,56505	1,67349	3,9	3,32544	1,5252	1,4981	2,2746	4,605	13,82
107	p107r5s6t7	1,38766	1,63982	3,98129	2,7672	1,5252	1,4981	2,2746	4,605	13,82
109	p109r5s6t7	1,99381	1,67562	3,3624	3,48314	1,5252	1,4981	2,2746	4,605	13,82
113	p113r5s6t7	3,23752	2,81502	4,01711	3,25672	1,5252	1,4981	2,2746	4,605	13,82
127	p127r5s6t7	1,47961	1,82369	2,38393	2,25596	1,5252	1,4981	2,2746	4,605	13,82
131	p131r5s6t7	1,59867	1,73316	3,64333	3,533	1,5252	1,4981	2,2746	4,605	13,82
137	p137r5s6t7	0,75472	2,16032	3,79629	3,71806	1,5252	1,4981	2,2746	4,605	13,82
139	p139r5s6t7	1,74533	1,44893	1,60334	2,39874	1,5252	1,4981	2,2746	4,605	13,82
149	p149r5s6t7	1,43489	2,66418	1,39581	1,27676	1,5252	1,4981	2,2746	4,605	13,82
151	p151r5s6t7	2,64499	1,95821	2,03937	1,98393	1,5252	1,4981	2,2746	4,605	13,82
157	p157r5s6t7	2,7461	2,41688	3,85116	3,62744	1,5252	1,4981	2,2746	4,605	13,82
163	p163r5s6t7	1,67622	1,85767	3,09737	2,64386	1,5252	1,4981	2,2746	4,605	13,82
167	p167r5s6t7	2,5277	2,74745	3,65904	4,50323	1,5252	1,4981	2,2746	4,605	13,82
173	p173r4s5t6	1,50882	1,6802	4,2646	3,31066	1,5252	1,4981	2,2746	4,605	13,82
179	p179r5s6t7	3,07321	2,79632	2,11382	3,00046	1,5252	1,4981	2,2746	4,605	13,82
181	p181r5s6t7	1,83214	2,18095	2,63843	2,54816	1,5252	1,4981	2,2746	4,605	13,82
191	p191r5s6t7	1,36426	1,46486	2,86868	4,07844	1,5252	1,4981	2,2746	4,605	13,82
193	p193r5s6t7	2,09208	1,04967	3,00253	1,86218	1,5252	1,4981	2,2746	4,605	13,82
197	p197r5s6t7	2,93082	2,22587	2,07117	1,83111	1,5252	1,4981	2,2746	4,605	13,82
199	p199r5s6t7	1,89799	2,0061	3,99876	3,1032	1,5252	1,4981	2,2746	4,605	13,82
211	p211r5s6t7	3,91416	2,96734	3,1431	3,16195	1,5252	1,4981	2,2746	4,605	13,82
223	p223r5s6t7	1,99632	2,06121	2,89193	3,33416	1,5252	1,4981	2,2746	4,605	13,82
227	p227r5s6t7	2,69208	1,77425	2,29306	1,66847	1,5252	1,4981	2,2746	4,605	13,82
229	p229r5s6t7	1,4256	0,8893	1,93306	2,15112	1,5252	1,4981	2,2746	4,605	13,82
233	p233r5s6t7	1,24754	2,22525	2,05203	1,91551	1,5252	1,4981	2,2746	4,605	13,82
239	p239r5s6t7	1,61983	2,28621	2,1511	1,84004	1,5252	1,4981	2,2746	4,605	13,82
241	p241r5s6t7	2,81522	2,24339	4,31214	4,69298	1,5252	1,4981	2,2746	4,605	13,82
251	p251r5s6t7	1,97388	1,2967	1,65857	1,4346	1,5252	1,4981	2,2746	4,605	13,82
257	p257r4s5t7	2,55257	2,04432	3,35572	1,92446	1,5252	1,4981	2,2746	4,605	13,82
263	p263r5s6t7	2,58818	2,68588	3,38912	2,88024	1,5252	1,4981	2,2746	4,605	13,82
269	p269r5s6t7	2,04443	2,00957	2,22182	2,3514	1,5252	1,4981	2,2746	4,605	13,82
271	p271r4s5t6	3,76695	2,00911	1,62482	1,69838	1,5252	1,4981	2,2746	4,605	13,82
277	p277r5s6t7	1,66406	1,39189	1,49833	3,95147	1,5252	1,4981	2,2746	4,605	13,82
281	p281r5s6t7	3,29937	2,27775	2,78648	1,91865	1,5252	1,4981	2,2746	4,605	13,82
283	p283r5s6t7	1,46473	2,70119	3,16386	1,43378	1,5252	1,4981	2,2746	4,605	13,82
293	p293r4s6t7	1,51193	2,26364	2,1207	2,91508	1,5252	1,4981	2,2746	4,605	13,82
307	p307r5s6t7	1,4193	1,73309	1,40606	1,30462	1,5252	1,4981	2,2746	4,605	13,82
311	p311r5s6t7	2,1168	2,04266	1,08686	0,9547	1,5252	1,4981	2,2746	4,605	13,82
313	p313r5s6t7	2,38588	2,46653	2,48797	1,94628	1,5252	1,4981	2,2746	4,605	13,82

Tabla del test serial para combinaciones *min*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	1,17861	1,66443	2397,91967	2171,79182	1,5252	1,4981	2,2746	4,605	13,82
5	p5r3s5t6	2,10976	1,84753	842,25855	808,67424	1,5252	1,4981	2,2746	4,605	13,82
7	p7r3s4t5	2,25901	2,23947	430,04565	423,28103	1,5252	1,4981	2,2746	4,605	13,82
11	p11r2s4t5	2,27287	1,08276	162,55367	161,92008	1,5252	1,4981	2,2746	4,605	13,82
13	p13r2s4t5	1,02761	1,68605	128,39164	126,71809	1,5252	1,4981	2,2746	4,605	13,82
17	p17r2s4t5	3,39437	3,04666	69,86828	71,00226	1,5252	1,4981	2,2746	4,605	13,82
19	p19r2s4t5	1,64246	1,58489	52,50207	50,77397	1,5252	1,4981	2,2746	4,605	13,82
23	p23r2s3t5	2,52753	2,32208	43,23494	42,41721	1,5252	1,4981	2,2746	4,605	13,82
29	p29r2s3t5	2,72965	2,97449	23,2417	23,19152	1,5252	1,4981	2,2746	4,605	13,82
31	p31r2s3t4	1,48587	1,09783	26,18547	25,67492	1,5252	1,4981	2,2746	4,605	13,82
37	p37r2s3t4	1,76774	1,61139	17,12584	17,77341	1,5252	1,4981	2,2746	4,605	13,82
41	p41r2s3t4	1,36006	1,03036	13,54098	13,19514	1,5252	1,4981	2,2746	4,605	13,82
43	p43r2s3t4	1,33546	1,62218	11,27075	11,79955	1,5252	1,4981	2,2746	4,605	13,82
47	p47r2s3t4	2,29231	2,10055	8,53803	7,9886	1,5252	1,4981	2,2746	4,605	13,82
53	p53r2s3t4	1,82099	0,87744	7,49893	7,65459	1,5252	1,4981	2,2746	4,605	13,82
59	p59r2s3t4	1,62473	1,72306	8,88038	9,18275	1,5252	1,4981	2,2746	4,605	13,82
61	p61r2s3t4	1,44491	1,34471	5,67264	5,86753	1,5252	1,4981	2,2746	4,605	13,82
67	p67r2s3t4	2,75077	3,36788	5,76299	6,33536	1,5252	1,4981	2,2746	4,605	13,82
71	p71r2s3t4	2,07008	3,31095	5,88005	5,66181	1,5252	1,4981	2,2746	4,605	13,82
73	p73r2s3t4	1,43696	1,69938	4,61581	4,9804	1,5252	1,4981	2,2746	4,605	13,82
79	p79r2s3t4	1,82116	1,83397	6,20947	5,42143	1,5252	1,4981	2,2746	4,605	13,82
83	p83r2s3t4	1,63476	2,21259	6,2607	6,33701	1,5252	1,4981	2,2746	4,605	13,82
89	p89r2s3t4	1,60614	1,82777	1,45914	1,06678	1,5252	1,4981	2,2746	4,605	13,82
97	p97r2s3t4	2,34493	2,23138	6,72991	4,98363	1,5252	1,4981	2,2746	4,605	13,82
101	p101r2s3t4	1,80137	2,46571	3,34033	3,59886	1,5252	1,4981	2,2746	4,605	13,82
103	p103r2s3t4	1,25863	2,24379	3,63314	4,22165	1,5252	1,4981	2,2746	4,605	13,82
107	p107r1s3t4	2,48987	2,07913	4,45076	3,41999	1,5252	1,4981	2,2746	4,605	13,82
109	p109r1s3t4	3,79399	2,1469	4,3501	4,24026	1,5252	1,4981	2,2746	4,605	13,82
113	p113r1s3t4	1,51538	1,88085	3,19781	4,01231	1,5252	1,4981	2,2746	4,605	13,82
127	p127r1s3t4	1,4273	1,74805	4,30001	3,33564	1,5252	1,4981	2,2746	4,605	13,82
131	p131r1s3t4	2,59594	2,42067	3,10792	3,52653	1,5252	1,4981	2,2746	4,605	13,82
137	p137r1s3t4	1,97105	1,77045	2,22821	2,55483	1,5252	1,4981	2,2746	4,605	13,82
139	p139r1s3t4	1,02941	1,55504	2,58162	2,66758	1,5252	1,4981	2,2746	4,605	13,82
149	p149r1s3t4	3,68884	3,38448	3,20581	3,30733	1,5252	1,4981	2,2746	4,605	13,82
151	p151r1s3t4	2,10501	1,37128	4,04979	4,39144	1,5252	1,4981	2,2746	4,605	13,82
157	p157r1s3t4	3,38719	1,56348	1,83512	1,9004	1,5252	1,4981	2,2746	4,605	13,82
163	p163r1s3t4	1,48428	1,33175	2,10722	2,19836	1,5252	1,4981	2,2746	4,605	13,82
167	p167r1s3t4	1,86747	1,44368	2,21409	1,37226	1,5252	1,4981	2,2746	4,605	13,82
173	p173r1s3t4	1,59257	1,71736	2,66648	3,11749	1,5252	1,4981	2,2746	4,605	13,82
179	p179r1s3t4	2,22622	2,80656	3,52371	3,60912	1,5252	1,4981	2,2746	4,605	13,82
181	p181r1s3t4	1,51676	1,18648	3,90067	2,47372	1,5252	1,4981	2,2746	4,605	13,82
191	p191r1s3t4	0,97545	0,9982	1,71857	2,09522	1,5252	1,4981	2,2746	4,605	13,82
193	p193r1s3t4	1,68687	2,17297	1,84913	2,07062	1,5252	1,4981	2,2746	4,605	13,82
197	p197r1s3t4	2,02311	2,03831	2,83566	3,04004	1,5252	1,4981	2,2746	4,605	13,82
199	p199r1s3t4	3,08963	2,46278	2,87736	2,77151	1,5252	1,4981	2,2746	4,605	13,82
211	p211r1s3t4	2,60163	2,45556	2,65839	3,789	1,5252	1,4981	2,2746	4,605	13,82
223	p223r1s3t4	1,35213	0,97841	1,41684	0,91464	1,5252	1,4981	2,2746	4,605	13,82
227	p227r1s3t4	2,50816	2,19365	2,47047	2,06607	1,5252	1,4981	2,2746	4,605	13,82
229	p229r1s3t4	3,13111	2,08462	2,55312	1,58883	1,5252	1,4981	2,2746	4,605	13,82
233	p233r1s3t4	2,32811	1,99077	1,98674	2,32536	1,5252	1,4981	2,2746	4,605	13,82
239	p239r1s3t4	1,76424	2,11762	2,19532	2,92664	1,5252	1,4981	2,2746	4,605	13,82
241	p241r1s3t4	2,30697	2,58486	2,14128	1,74106	1,5252	1,4981	2,2746	4,605	13,82
251	p251r1s3t4	1,62381	0,95244	3,13905	2,97728	1,5252	1,4981	2,2746	4,605	13,82
257	p257r1s3t4	1,77607	1,20377	2,12925	2,69655	1,5252	1,4981	2,2746	4,605	13,82
263	p263r1s3t4	2,23575	2,23841	1,83005	2,0864	1,5252	1,4981	2,2746	4,605	13,82
269	p269r1s3t4	3,0437	2,30924	1,77912	1,22685	1,5252	1,4981	2,2746	4,605	13,82
271	p271r1s3t4	2,0619	1,78569	2,54747	2,80143	1,5252	1,4981	2,2746	4,605	13,82
277	p277r1s3t4	2,27451	2,83343	1,9661	1,91219	1,5252	1,4981	2,2746	4,605	13,82
281	p281r1s3t4	1,70428	2,12997	2,00528	2,87107	1,5252	1,4981	2,2746	4,605	13,82
283	p283r1s3t4	1,39998	1,25536	2,62815	2,86865	1,5252	1,4981	2,2746	4,605	13,82
293	p293r1s3t4	3,56453	2,44656	2,51367	2,7932	1,5252	1,4981	2,2746	4,605	13,82
307	p307r1s3t4	2,21974	1,31233	2,17701	1,97358	1,5252	1,4981	2,2746	4,605	13,82
311	p311r1s3t4	1,59199	2,37846	2,93754	2,60435	1,5252	1,4981	2,2746	4,605	13,82
313	p313r1s3t4	1,59201	2,24644	3,01938	3,98778	1,5252	1,4981	2,2746	4,605	13,82

Tabla del test serial para combinaciones *bottom*.

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	1,17861	1,66443	2397,91967	2171,79182	1,5252	1,4981	2,2746	4,605	13,82
5	p5r2s5t7	2,2532	1,93409	828,94107	799,37392	1,5252	1,4981	2,2746	4,605	13,82
7	p7r1s5t7	1,73129	2,53983	435,2037	429,76253	1,5252	1,4981	2,2746	4,605	13,82
11	p11r1s4t7	1,08927	2,16659	165,07639	163,23875	1,5252	1,4981	2,2746	4,605	13,82
13	p13r1s5t6	2,04079	1,55731	105,34871	106,11999	1,5252	1,4981	2,2746	4,605	13,82
17	p17r1s4t6	1,19664	1,39364	72,39389	72,24553	1,5252	1,4981	2,2746	4,605	13,82
19	p19r1s3t7	3,00353	0,85187	55,76247	55,29137	1,5252	1,4981	2,2746	4,605	13,82
23	p23r1s3t7	1,93813	1,7998	41,58763	42,52727	1,5252	1,4981	2,2746	4,605	13,82
29	p29r1s3t7	2,50033	2,91896	23,72909	23,1169	1,5252	1,4981	2,2746	4,605	13,82
31	p31r1s3t7	2,43272	1,94666	19,85471	20,07329	1,5252	1,4981	2,2746	4,605	13,82
37	p37r1s3t7	2,05854	1,34807	17,75339	18,13048	1,5252	1,4981	2,2746	4,605	13,82
41	p41r1s3t7	1,84158	1,53647	11,32813	11,27405	1,5252	1,4981	2,2746	4,605	13,82
43	p43r1s3t7	1,02373	1,02143	21,0945	21,41257	1,5252	1,4981	2,2746	4,605	13,82
47	p47r1s3t7	1,73206	1,48831	12,10957	12,48242	1,5252	1,4981	2,2746	4,605	13,82
53	p53r1s2t7	2,10453	2,1083	7,66614	7,07521	1,5252	1,4981	2,2746	4,605	13,82
59	p59r1s3t5	2,19764	2,68574	7,34158	7,12799	1,5252	1,4981	2,2746	4,605	13,82
61	p61r1s2t7	1,87552	1,98354	9,21073	11,27579	1,5252	1,4981	2,2746	4,605	13,82
67	p67r1s3t5	2,16756	1,89125	5,86473	5,93366	1,5252	1,4981	2,2746	4,605	13,82
71	p71r1s3t5	1,55526	2,13399	2,23605	3,24522	1,5252	1,4981	2,2746	4,605	13,82
73	p73r1s2t7	2,03355	1,45767	6,20343	5,28602	1,5252	1,4981	2,2746	4,605	13,82
79	p79r1s3t5	2,8584	3,61764	5,40037	5,03725	1,5252	1,4981	2,2746	4,605	13,82
83	p83r1s3t5	1,93685	1,93791	5,38477	4,39474	1,5252	1,4981	2,2746	4,605	13,82
89	p89r1s3t5	1,38138	1,66098	5,49196	5,25484	1,5252	1,4981	2,2746	4,605	13,82
97	p97r1s2t7	1,84213	1,81385	1,68422	1,62138	1,5252	1,4981	2,2746	4,605	13,82
101	p101r1s3t5	2,5121	1,54705	2,53144	4,26673	1,5252	1,4981	2,2746	4,605	13,82
103	p103r1s2t7	1,36563	2,15484	2,32137	2,91105	1,5252	1,4981	2,2746	4,605	13,82
107	p107r1s3t4	2,48987	2,07913	4,45076	3,41999	1,5252	1,4981	2,2746	4,605	13,82
109	p109r1s3t4	3,79399	2,1469	4,3501	4,24026	1,5252	1,4981	2,2746	4,605	13,82
113	p113r1s3t4	1,51538	1,88085	3,19781	4,01231	1,5252	1,4981	2,2746	4,605	13,82
127	p127r1s3t4	1,4273	1,74805	4,30001	3,33564	1,5252	1,4981	2,2746	4,605	13,82
131	p131r1s3t4	2,59594	2,42067	3,10792	3,52653	1,5252	1,4981	2,2746	4,605	13,82
137	p137r1s2t7	1,97219	1,80325	2,28552	2,40037	1,5252	1,4981	2,2746	4,605	13,82
139	p139r1s3t4	1,02941	1,55504	2,58162	2,66758	1,5252	1,4981	2,2746	4,605	13,82
149	p149r1s3t4	3,68884	3,38448	3,20581	3,30733	1,5252	1,4981	2,2746	4,605	13,82
151	p151r1s2t7	1,34302	1,16072	4,33006	4,33241	1,5252	1,4981	2,2746	4,605	13,82
157	p157r1s2t7	1,15801	1,05704	3,78418	4,23022	1,5252	1,4981	2,2746	4,605	13,82
163	p163r1s2t7	1,67749	2,61229	2,9838	2,21165	1,5252	1,4981	2,2746	4,605	13,82
167	p167r1s3t4	1,86747	1,44368	2,21409	1,37226	1,5252	1,4981	2,2746	4,605	13,82
173	p173r1s3t4	1,59257	1,71736	2,66648	3,11749	1,5252	1,4981	2,2746	4,605	13,82
179	p179r1s3t4	2,22622	2,80656	3,52371	3,60912	1,5252	1,4981	2,2746	4,605	13,82
181	p181r1s3t4	1,51676	1,18648	3,90067	2,47372	1,5252	1,4981	2,2746	4,605	13,82
191	p191r1s3t4	0,97545	0,9982	1,71857	2,09522	1,5252	1,4981	2,2746	4,605	13,82
193	p193r1s2t5	1,72117	2,20054	2,84044	2,70679	1,5252	1,4981	2,2746	4,605	13,82
197	p197r1s3t4	2,02311	2,03831	2,83566	3,04004	1,5252	1,4981	2,2746	4,605	13,82
199	p199r1s3t4	3,08963	2,46278	2,87736	2,77151	1,5252	1,4981	2,2746	4,605	13,82
211	p211r1s2t5	2,22168	2,05473	3,87881	4,06802	1,5252	1,4981	2,2746	4,605	13,82
223	p223r1s3t4	1,35213	0,97841	1,41684	0,91464	1,5252	1,4981	2,2746	4,605	13,82
227	p227r1s3t4	2,50816	2,19365	2,47047	2,06607	1,5252	1,4981	2,2746	4,605	13,82
229	p229r1s2t5	2,52377	2,21706	1,46395	1,55416	1,5252	1,4981	2,2746	4,605	13,82
233	p233r1s3t4	2,32811	1,99077	1,98674	2,32536	1,5252	1,4981	2,2746	4,605	13,82
239	p239r1s3t4	1,76424	2,11762	2,19532	2,92664	1,5252	1,4981	2,2746	4,605	13,82
241	p241r1s2t5	2,07165	3,21964	1,72549	2,18144	1,5252	1,4981	2,2746	4,605	13,82
251	p251r1s3t4	1,62381	0,95244	3,13905	2,97728	1,5252	1,4981	2,2746	4,605	13,82
257	p257r1s2t5	1,95936	1,58103	2,10276	2,07937	1,5252	1,4981	2,2746	4,605	13,82
263	p263r1s2t5	2,9018	2,28352	1,93594	3,15446	1,5252	1,4981	2,2746	4,605	13,82
269	p269r1s3t4	3,0437	2,30924	1,77912	1,22685	1,5252	1,4981	2,2746	4,605	13,82
271	p271r1s2t5	2,08296	2,14255	2,66916	2,89858	1,5252	1,4981	2,2746	4,605	13,82
277	p277r1s2t5	2,53493	2,67866	1,90726	2,4995	1,5252	1,4981	2,2746	4,605	13,82
281	p281r1s2t5	2,63828	1,24289	1,89828	1,39136	1,5252	1,4981	2,2746	4,605	13,82
283	p283r1s2t5	2,13345	1,33061	2,42665	1,88422	1,5252	1,4981	2,2746	4,605	13,82
293	p293r1s2t5	3,3884	2,35942	1,74156	2,34128	1,5252	1,4981	2,2746	4,605	13,82
307	p307r1s3t4	2,21974	1,31233	2,17701	1,97358	1,5252	1,4981	2,2746	4,605	13,82
311	p311r1s3t4	1,59199	2,37846	2,93754	2,60435	1,5252	1,4981	2,2746	4,605	13,82
313	p313r1s2t5	1,91843	1,66165	2,86373	2,819	1,5252	1,4981	2,2746	4,605	13,82

Tabla del test de póker 8 para combinaciones *top*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r5s6t7	260,008	267,585	3413,45666	2881,35	249	254,91	276,35	284,3	330,5
5	p5r5s6t7	256,321	260,212	1167,16928	1048,84	249	254,91	276,35	284,3	330,5
7	p7r5s6t7	253,597	249,399	691,82848	639,011	249	254,91	276,35	284,3	330,5
11	p11r5s6t7	261,851	261,134	430,31936	412,87	249	254,91	276,35	284,3	330,5
13	p13r5s6t7	252,307	263,981	396,9984	364,804	249	254,91	276,35	284,3	330,5
17	p17r5s6t7	249,44	250,525	330,31552	326,015	249	254,91	276,35	284,3	330,5
19	p19r5s6t7	259,127	259,721	319,25632	310,429	249	254,91	276,35	284,3	330,5
23	p23r5s6t7	255,932	245,815	302,176	289,704	249	254,91	276,35	284,3	330,5
29	p29r5s6t7	258,41	250,894	276,12544	274,385	249	254,91	276,35	284,3	330,5
31	p31r5s6t7	259,209	254,54	286,304	287,942	249	254,91	276,35	284,3	330,5
37	p37r5s6t7	245,979	249,399	254,21184	263,551	249	254,91	276,35	284,3	330,5
41	p41r5s6t7	255,707	257,591	275,49056	254,806	249	254,91	276,35	284,3	330,5
43	p43r5s6t7	255,318	247,453	262,42432	271,354	249	254,91	276,35	284,3	330,5
47	p47r5s6t7	249,03	245,487	259,35232	256,71	249	254,91	276,35	284,3	330,5
53	p53r5s6t7	249,645	257,734	262,62912	275,286	249	254,91	276,35	284,3	330,5
59	p59r5s6t7	267,565	257,489	261,85088	261,011	249	254,91	276,35	284,3	330,5
61	p61r5s6t7	264,657	248,58	248,84608	257,652	249	254,91	276,35	284,3	330,5
67	p67r5s6t7	250,321	246,388	253,35168	259,803	249	254,91	276,35	284,3	330,5
71	p71r5s6t7	238,483	246,204	252,9216	249,01	249	254,91	276,35	284,3	330,5
73	p73r5s6t7	263,448	258,062	256,68992	266,418	249	254,91	276,35	284,3	330,5
79	p79r5s6t7	266,561	275,839	242,66112	245,59	249	254,91	276,35	284,3	330,5
83	p83r5s6t7	248,846	249,706	264,57472	257,243	249	254,91	276,35	284,3	330,5
89	p89r5s6t7	254,294	249,051	267,03232	262,732	249	254,91	276,35	284,3	330,5
97	p97r5s6t7	258,554	245,631	247,92448	259,619	249	254,91	276,35	284,3	330,5
101	p101r5s6t7	235,391	253,352	252,65536	244,525	249	254,91	276,35	284,3	330,5
103	p103r5s6t7	256,547	252,778	254,53952	240,777	249	254,91	276,35	284,3	330,5
107	p107r5s6t7	270,678	273,197	254,33472	253,7	249	254,91	276,35	284,3	330,5
109	p109r5s6t7	260,765	246,655	250,464	250,157	249	254,91	276,35	284,3	330,5
113	p113r5s6t7	265,804	254,642	246,85952	260,806	249	254,91	276,35	284,3	330,5
127	p127r5s6t7	246,04	242,067	264,288	257,775	249	254,91	276,35	284,3	330,5
131	p131r5s6t7	268,732	257,878	258,38976	254,109	249	254,91	276,35	284,3	330,5
137	p137r5s6t7	251,734	249,235	255,54304	261,462	249	254,91	276,35	284,3	330,5
139	p139r5s6t7	261,769	253,638	266,09024	265,537	249	254,91	276,35	284,3	330,5
149	p149r5s6t7	261,564	253,802	255,97312	260,909	249	254,91	276,35	284,3	330,5
151	p151r5s6t7	258,656	259,782	254,29376	244,238	249	254,91	276,35	284,3	330,5
157	p157r5s6t7	256,424	253,577	260,82688	255,83	249	254,91	276,35	284,3	330,5
163	p163r5s6t7	266,52	260,929	251,7952	259,803	249	254,91	276,35	284,3	330,5
167	p167r5s6t7	254,683	248,006	252,24576	255,564	249	254,91	276,35	284,3	330,5
173	p173r4s5t6	249,809	255,584	268,2816	256,567	249	254,91	276,35	284,3	330,5
179	p179r5s6t7	259,332	258,451	252,24576	244,361	249	254,91	276,35	284,3	330,5
181	p181r5s6t7	245,856	254,396	256,05504	249,85	249	254,91	276,35	284,3	330,5
191	p191r5s6t7	256,444	257,919	246,28608	250,484	249	254,91	276,35	284,3	330,5
193	p193r5s6t7	264,001	257,181	263,18208	251,898	249	254,91	276,35	284,3	330,5
197	p197r5s6t7	254,929	248,6	259,61856	255,625	249	254,91	276,35	284,3	330,5
199	p199r5s6t7	264,349	256,813	251,40608	249,87	249	254,91	276,35	284,3	330,5
211	p211r5s6t7	251,57	250,362	249,82912	253,597	249	254,91	276,35	284,3	330,5
223	p223r5s6t7	253,843	249,87	260,25344	250,218	249	254,91	276,35	284,3	330,5
227	p227r5s6t7	258,595	254,785	243,35744	237,377	249	254,91	276,35	284,3	330,5
229	p229r5s6t7	244,074	240,306	259,61856	259,229	249	254,91	276,35	284,3	330,5
233	p233r5s6t7	257,632	243,419	257,38624	254,929	249	254,91	276,35	284,3	330,5
239	p239r5s6t7	258,001	261,81	265,6192	260,438	249	254,91	276,35	284,3	330,5
241	p241r5s6t7	260,028	256,342	267,70816	263,366	249	254,91	276,35	284,3	330,5
251	p251r5s6t7	257,857	259,373	249,05088	251,078	249	254,91	276,35	284,3	330,5
257	p257r4s5t7	246,737	258,083	258,12352	249,624	249	254,91	276,35	284,3	330,5
263	p263r5s6t7	265,025	261,257	258,53312	250,136	249	254,91	276,35	284,3	330,5
269	p269r5s6t7	237,275	250,136	259,51616	258,615	249	254,91	276,35	284,3	330,5
271	p271r4s5t6	242,538	242,415	243,9104	228,387	249	254,91	276,35	284,3	330,5
277	p277r5s6t7	252,41	259,844	256,83328	253,311	249	254,91	276,35	284,3	330,5
281	p281r5s6t7	261,769	256,649	261,25696	257,714	249	254,91	276,35	284,3	330,5
283	p283r5s6t7	255,953	256,28	262,67008	252,225	249	254,91	276,35	284,3	330,5
293	p293r4s6t7	255,83	264,718	258,69696	259,987	249	254,91	276,35	284,3	330,5
307	p307r5s6t7	257,898	262,015	245,05728	245,999	249	254,91	276,35	284,3	330,5
311	p311r5s6t7	252,635	266,336	260,64256	264,984	249	254,91	276,35	284,3	330,5
313	p313r5s6t7	253,72	249,297	259,3728	254,191	249	254,91	276,35	284,3	330,5

Tabla del test de póker 8 para combinaciones *min*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	256,116	254,519	3441,71906	2921,63	249	254,91	276,35	284,3	330,5
5	p5r3s5t6	268,691	262,793	1177,32736	1054	249	254,91	276,35	284,3	330,5
7	p7r3s4t5	264,165	270,534	706,08256	639,83	249	254,91	276,35	284,3	330,5
11	p11r2s4t5	252,799	255,052	420,01792	400,214	249	254,91	276,35	284,3	330,5
13	p13r2s4t5	247,556	255,932	392,5952	358,086	249	254,91	276,35	284,3	330,5
17	p17r2s4t5	247,453	250,341	321,12	307,726	249	254,91	276,35	284,3	330,5
19	p19r2s4t5	261,667	255,072	316,88064	301,992	249	254,91	276,35	284,3	330,5
23	p23r2s3t5	245,938	255,215	297,83424	303,2	249	254,91	276,35	284,3	330,5
29	p29r2s3t5	242,641	247,535	284,44032	269,08	249	254,91	276,35	284,3	330,5
31	p31r2s3t4	254,314	258,308	275,28576	271,866	249	254,91	276,35	284,3	330,5
37	p37r2s3t4	246,757	256,28	266,76608	265,783	249	254,91	276,35	284,3	330,5
41	p41r2s3t4	249,686	253,925	267,60576	260,028	249	254,91	276,35	284,3	330,5
43	p43r2s3t4	254,478	248,15	253,29024	256,813	249	254,91	276,35	284,3	330,5
47	p47r2s3t4	255,174	263,776	261,78944	257,53	249	254,91	276,35	284,3	330,5
53	p53r2s3t4	250,853	241,473	255,05152	252,225	249	254,91	276,35	284,3	330,5
59	p59r2s3t4	252,451	255,645	263,01824	244,893	249	254,91	276,35	284,3	330,5
61	p61r2s3t4	245,59	243,316	273,504	269,777	249	254,91	276,35	284,3	330,5
67	p67r2s3t4	245,283	249,583	252,34816	260,786	249	254,91	276,35	284,3	330,5
71	p71r2s3t4	253,864	251,508	290,62528	261,708	249	254,91	276,35	284,3	330,5
73	p73r2s3t4	252,614	248,109	256,15744	251,713	249	254,91	276,35	284,3	330,5
79	p79r2s3t4	250,362	251,201	262,6496	271,886	249	254,91	276,35	284,3	330,5
83	p83r2s3t4	252,307	247,494	247,10528	248,498	249	254,91	276,35	284,3	330,5
89	p89r2s3t4	254,519	250,771	246,5728	249,645	249	254,91	276,35	284,3	330,5
97	p97r2s3t4	252,778	262,732	266,17216	267,77	249	254,91	276,35	284,3	330,5
101	p101r2s3t4	249,972	250,013	264,9024	252,983	249	254,91	276,35	284,3	330,5
103	p103r2s3t4	252,819	259,373	260,33536	261,871	249	254,91	276,35	284,3	330,5
107	p107r1s3t4	262,854	258,328	254,29376	253,946	249	254,91	276,35	284,3	330,5
109	p109r1s3t4	256,547	248,723	252,79872	250,648	249	254,91	276,35	284,3	330,5
113	p113r1s3t4	259,762	248,621	243,84896	250,956	249	254,91	276,35	284,3	330,5
127	p127r1s3t4	251,59	259,721	242,6816	243,726	249	254,91	276,35	284,3	330,5
131	p131r1s3t4	261,564	245,631	267,89248	263,121	249	254,91	276,35	284,3	330,5
137	p137r1s3t4	257,489	265,885	257,87776	263,715	249	254,91	276,35	284,3	330,5
139	p139r1s3t4	247,781	256,28	248,96896	273,32	249	254,91	276,35	284,3	330,5
149	p149r1s3t4	258,84	251,16	259,08608	262,772	249	254,91	276,35	284,3	330,5
151	p151r1s3t4	255,625	245,856	253,00352	261,421	249	254,91	276,35	284,3	330,5
157	p157r1s3t4	250,996	248,928	257,55008	262,117	249	254,91	276,35	284,3	330,5
163	p163r1s3t4	253,024	248,6	254,12992	258,287	249	254,91	276,35	284,3	330,5
167	p167r1s3t4	260,684	260,335	247,74016	250,341	249	254,91	276,35	284,3	330,5
173	p173r1s3t4	262,895	260,52	260,58112	261,277	249	254,91	276,35	284,3	330,5
179	p179r1s3t4	264,595	257,55	255,3792	247,679	249	254,91	276,35	284,3	330,5
181	p181r1s3t4	252,246	252,492	266,9504	271,722	249	254,91	276,35	284,3	330,5
191	p191r1s3t4	258,185	260,09	250,60736	255,604	249	254,91	276,35	284,3	330,5
193	p193r1s3t4	249,645	250,669	251,87712	254,109	249	254,91	276,35	284,3	330,5
197	p197r1s3t4	258,083	254,847	250,13632	248,498	249	254,91	276,35	284,3	330,5
199	p199r1s3t4	249,932	254,826	250,5664	254,58	249	254,91	276,35	284,3	330,5
211	p211r1s3t4	250,321	260,335	242,4768	252,532	249	254,91	276,35	284,3	330,5
223	p223r1s3t4	253,29	243,48	264,9024	248,498	249	254,91	276,35	284,3	330,5
227	p227r1s3t4	261,421	258,246	258,79936	250,607	249	254,91	276,35	284,3	330,5
229	p229r1s3t4	267,319	258,369	256,42368	255,011	249	254,91	276,35	284,3	330,5
233	p233r1s3t4	259,107	254,765	237,9712	253,249	249	254,91	276,35	284,3	330,5
239	p239r1s3t4	247,146	247,208	265,45536	255,338	249	254,91	276,35	284,3	330,5
241	p241r1s3t4	256,833	253,044	259,2704	257,612	249	254,91	276,35	284,3	330,5
251	p251r1s3t4	248,744	249,276	243,76704	241,699	249	254,91	276,35	284,3	330,5
257	p257r1s3t4	236,374	243,276	265,04576	259,782	249	254,91	276,35	284,3	330,5
263	p263r1s3t4	249,522	249,604	256,52608	256,035	249	254,91	276,35	284,3	330,5
269	p269r1s3t4	259,004	259,639	252,45056	253,27	249	254,91	276,35	284,3	330,5
271	p271r1s3t4	243,316	239,364	261,91232	261,585	249	254,91	276,35	284,3	330,5
277	p277r1s3t4	264,39	271,128	261,56416	253,085	249	254,91	276,35	284,3	330,5
281	p281r1s3t4	242,907	248,355	261,87136	253,925	249	254,91	276,35	284,3	330,5
283	p283r1s3t4	253,556	259,885	275,30624	256,178	249	254,91	276,35	284,3	330,5
293	p293r1s3t4	254,253	252,205	264,10368	257,857	249	254,91	276,35	284,3	330,5
307	p307r1s3t4	244,852	241,617	257,03808	261,585	249	254,91	276,35	284,3	330,5
311	p311r1s3t4	257,468	262,097	242,96832	246,921	249	254,91	276,35	284,3	330,5
313	p313r1s3t4	252	256,116	252,9216	251,345	249	254,91	276,35	284,3	330,5

Tabla del test de póker 8 para combinaciones *bottom*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	256,116	254,519	3441,71906	2921,63	249	254,91	276,35	284,3	330,5
5	p5r2s5t7	255,83	247,023	1180,5837	1047,61	249	254,91	276,35	284,3	330,5
7	p7r1s5t7	248,129	251,386	738,58432	667,887	249	254,91	276,35	284,3	330,5
11	p11r1s4t7	256,772	256,035	439,55584	423,254	249	254,91	276,35	284,3	330,5
13	p13r1s5t6	257,98	256,936	356,93952	358,578	249	254,91	276,35	284,3	330,5
17	p17r1s4t6	252,266	245,549	324,56064	318,847	249	254,91	276,35	284,3	330,5
19	p19r1s3t7	257,55	255,256	300,61952	306,395	249	254,91	276,35	284,3	330,5
23	p23r1s3t7	252,164	245,979	300,92672	294,435	249	254,91	276,35	284,3	330,5
29	p29r1s3t7	263,653	256,956	282,02368	269,838	249	254,91	276,35	284,3	330,5
31	p31r1s3t7	256,465	257,734	273,11488	272,029	249	254,91	276,35	284,3	330,5
37	p37r1s3t7	255,707	256,116	258,81984	263,284	249	254,91	276,35	284,3	330,5
41	p41r1s3t7	257,571	254,56	264,53376	250,403	249	254,91	276,35	284,3	330,5
43	p43r1s3t7	253,372	257,652	278,46016	272,132	249	254,91	276,35	284,3	330,5
47	p47r1s3t7	247,515	251,939	253,18784	260,151	249	254,91	276,35	284,3	330,5
53	p53r1s2t7	261,175	260,54	259,5776	266,623	249	254,91	276,35	284,3	330,5
59	p59r1s3t5	259,578	271,538	251,61088	251,181	249	254,91	276,35	284,3	330,5
61	p61r1s2t7	255,953	251,672	254,8672	254,724	249	254,91	276,35	284,3	330,5
67	p67r1s3t5	251,037	251,672	271,64032	268,179	249	254,91	276,35	284,3	330,5
71	p71r1s3t5	265,496	256,731	246,40896	249,932	249	254,91	276,35	284,3	330,5
73	p73r1s2t7	252,696	250,034	269,92	260,888	249	254,91	276,35	284,3	330,5
79	p79r1s3t5	264,534	267,462	270,20672	266,643	249	254,91	276,35	284,3	330,5
83	p83r1s3t5	242,415	255,871	245,93792	255,215	249	254,91	276,35	284,3	330,5
89	p89r1s3t5	257,734	262,957	271,29216	252,799	249	254,91	276,35	284,3	330,5
97	p97r1s2t7	254,294	248,191	257,69344	261,667	249	254,91	276,35	284,3	330,5
101	p101r1s3t5	248,211	250,321	247,96544	248,744	249	254,91	276,35	284,3	330,5
103	p103r1s2t7	246,143	248,457	251,09888	239,753	249	254,91	276,35	284,3	330,5
107	p107r1s3t4	262,854	258,328	254,29376	253,946	249	254,91	276,35	284,3	330,5
109	p109r1s3t4	256,547	248,723	252,79872	250,648	249	254,91	276,35	284,3	330,5
113	p113r1s3t4	259,762	248,621	243,84896	250,956	249	254,91	276,35	284,3	330,5
127	p127r1s3t4	251,59	259,721	242,6816	243,726	249	254,91	276,35	284,3	330,5
131	p131r1s3t4	261,564	245,631	267,89248	263,121	249	254,91	276,35	284,3	330,5
137	p137r1s2t7	264,943	260,131	243,76704	256,219	249	254,91	276,35	284,3	330,5
139	p139r1s3t4	247,781	256,28	248,96896	273,32	249	254,91	276,35	284,3	330,5
149	p149r1s3t4	258,84	251,16	259,08608	262,772	249	254,91	276,35	284,3	330,5
151	p151r1s2t7	253,536	252,676	251,01696	261,052	249	254,91	276,35	284,3	330,5
157	p157r1s2t7	248,006	260,049	260,37632	260,253	249	254,91	276,35	284,3	330,5
163	p163r1s2t7	265,455	259,926	245,87648	249,563	249	254,91	276,35	284,3	330,5
167	p167r1s3t4	260,684	260,335	247,74016	250,341	249	254,91	276,35	284,3	330,5
173	p173r1s3t4	262,895	260,52	260,58112	261,277	249	254,91	276,35	284,3	330,5
179	p179r1s3t4	264,595	257,55	255,3792	247,679	249	254,91	276,35	284,3	330,5
181	p181r1s3t4	252,246	252,492	266,9504	271,722	249	254,91	276,35	284,3	330,5
191	p191r1s3t4	258,185	260,09	250,60736	255,604	249	254,91	276,35	284,3	330,5
193	p193r1s2t5	243,951	254,601	248,928	248,805	249	254,91	276,35	284,3	330,5
197	p197r1s3t4	258,083	254,847	250,13632	248,498	249	254,91	276,35	284,3	330,5
199	p199r1s3t4	249,932	254,826	250,5664	254,58	249	254,91	276,35	284,3	330,5
211	p211r1s2t5	251,488	246,102	262,0352	254,826	249	254,91	276,35	284,3	330,5
223	p223r1s3t4	253,29	243,48	264,9024	248,498	249	254,91	276,35	284,3	330,5
227	p227r1s3t4	261,421	258,246	258,79936	250,607	249	254,91	276,35	284,3	330,5
229	p229r1s2t5	252,963	247,412	248,88704	251,98	249	254,91	276,35	284,3	330,5
233	p233r1s3t4	259,107	254,765	237,9712	253,249	249	254,91	276,35	284,3	330,5
239	p239r1s3t4	247,146	247,208	265,45536	255,338	249	254,91	276,35	284,3	330,5
241	p241r1s2t5	258,226	263,469	245,28256	245,446	249	254,91	276,35	284,3	330,5
251	p251r1s3t4	248,744	249,276	243,76704	241,699	249	254,91	276,35	284,3	330,5
257	p257r1s2t5	240,347	244,422	249,8496	250,423	249	254,91	276,35	284,3	330,5
263	p263r1s2t5	264,206	256,628	258,67648	258,82	249	254,91	276,35	284,3	330,5
269	p269r1s3t4	259,004	259,639	252,45056	253,27	249	254,91	276,35	284,3	330,5
271	p271r1s2t5	251,078	252,102	252,1024	250,853	249	254,91	276,35	284,3	330,5
277	p277r1s2t5	244,73	246,388	266,11072	260,008	249	254,91	276,35	284,3	330,5
281	p281r1s2t5	242,804	236,456	249,35808	260,561	249	254,91	276,35	284,3	330,5
283	p283r1s2t5	257,673	252,532	254,62144	261,953	249	254,91	276,35	284,3	330,5
293	p293r1s2t5	261,277	256,465	259,68	256,035	249	254,91	276,35	284,3	330,5
307	p307r1s3t4	244,852	241,617	257,03808	261,585	249	254,91	276,35	284,3	330,5
311	p311r1s3t4	257,468	262,097	242,96832	246,921	249	254,91	276,35	284,3	330,5
313	p313r1s2t5	250,771	243,153	257,09952	250,444	249	254,91	276,35	284,3	330,5

Tabla del test de póker 16 para combinaciones *top*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	0,001
3	p3r5s6t7	65439,43361	65397,49102	71091,25783	70493,5711	65607	65650	65681	65999	66659
5	p5r5s6t7	65670,12031	65680,60664	66561,40939	66330,72344	65607	65650	65681	65999	66659
7	p7r5s6t7	65607,20547	65533,80509	66194,40782	66005,66485	65607	65650	65681	65999	66659
11	p11r5s6t7	65554,77619	65544,29102	65743,52148	65617,69063	65607	65650	65681	65999	66659
13	p13r5s6t7	65512,83322	65722,54922	65963,72188	65837,89376	65607	65650	65681	65999	66659
17	p17r5s6t7	65481,37658	65470,89063	65428,94767	65523,31876	65607	65650	65681	65999	66659
19	p19r5s6t7	65617,6914	65754,00588	65628,17695	65481,37736	65607	65650	65681	65999	66659
23	p23r5s6t7	65659,63476	65785,4625	65554,77695	65575,74806	65607	65650	65681	65999	66659
29	p29r5s6t7	65407,97617	65376,51837	65680,60548	65712,06407	65607	65650	65681	65999	66659
31	p31r5s6t7	65565,26211	65533,8047	65596,71992	65387,00469	65607	65650	65681	65999	66659
37	p37r5s6t7	65533,80546	65460,40471	65533,8043	65533,80508	65607	65650	65681	65999	66659
41	p41r5s6t7	65407,97541	65575,74883	65428,94805	65449,91915	65607	65650	65681	65999	66659
43	p43r5s6t7	65512,8336	65376,51875	65659,63399	65554,77618	65607	65650	65681	65999	66659
47	p47r5s6t7	65575,74845	65586,23397	65407,9758	65680,60664	65607	65650	65681	65999	66659
53	p53r5s6t7	65397,49064	65407,97618	65407,9754	65387,00393	65607	65650	65681	65999	66659
59	p59r5s6t7	65533,80586	65617,69181	65418,46133	65407,97657	65607	65650	65681	65999	66659
61	p61r5s6t7	65754,00625	65481,37578	65376,51835	65523,31994	65607	65650	65681	65999	66659
67	p67r5s6t7	65649,14806	65607,20586	65523,31954	65586,23399	65607	65650	65681	65999	66659
71	p71r5s6t7	65345,06173	65502,34806	65512,83323	65502,34805	65607	65650	65681	65999	66659
73	p73r5s6t7	65565,26289	65512,8336	65449,91955	65533,80585	65607	65650	65681	65999	66659
79	p79r5s6t7	65586,23398	65554,77618	65554,77696	65638,66252	65607	65650	65681	65999	66659
83	p83r5s6t7	65523,31993	65565,26213	65628,17735	65481,37696	65607	65650	65681	65999	66659
89	p89r5s6t7	65670,1203	65533,80548	65607,20665	65701,57813	65607	65650	65681	65999	66659
97	p97r5s6t7	65701,57657	65544,29063	65460,40353	65502,34767	65607	65650	65681	65999	66659
101	p101r5s6t7	65691,09219	65733,03594	65512,8336	65701,57773	65607	65650	65681	65999	66659
103	p103r5s6t7	65617,6914	65565,26251	65366,03283	65387,00392	65607	65650	65681	65999	66659
107	p107r5s6t7	65586,23399	65397,49101	65481,37656	65565,26211	65607	65650	65681	65999	66659
109	p109r5s6t7	65575,74844	65670,11992	65240,20391	65460,4043	65607	65650	65681	65999	66659
113	p113r5s6t7	65439,43321	65397,48986	65649,14805	65670,11992	65607	65650	65681	65999	66659
127	p127r5s6t7	65428,94728	65575,74843	65596,71993	65470,89024	65607	65650	65681	65999	66659
131	p131r5s6t7	65502,34806	65565,2625	65418,46172	65533,80547	65607	65650	65681	65999	66659
137	p137r5s6t7	65439,43204	65638,66406	65544,29025	65628,17696	65607	65650	65681	65999	66659
139	p139r5s6t7	65470,89025	65638,66249	65387,00431	65491,86174	65607	65650	65681	65999	66659
149	p149r5s6t7	65523,31913	65628,17657	65376,51915	65554,77657	65607	65650	65681	65999	66659
151	p151r5s6t7	65366,0332	65397,49024	65649,14922	65806,43594	65607	65650	65681	65999	66659
157	p157r5s6t7	65586,23438	65491,86134	65680,60586	65659,63437	65607	65650	65681	65999	66659
163	p163r5s6t7	65271,66053	65387,0043	65533,80508	65533,80511	65607	65650	65681	65999	66659
167	p167r5s6t7	65575,74844	65439,43321	65418,46213	65512,83359	65607	65650	65681	65999	66659
173	p173r4s5t6	65481,37659	65533,80547	65439,43321	65575,74767	65607	65650	65681	65999	66659
179	p179r5s6t7	65701,57734	65565,26289	65491,86211	65502,34768	65607	65650	65681	65999	66659
181	p181r5s6t7	65502,34766	65586,23397	65491,86133	65355,54765	65607	65650	65681	65999	66659
191	p191r5s6t7	65733,03362	65754,00585	65670,12033	65607,20468	65607	65650	65681	65999	66659
193	p193r5s6t7	65586,23321	65722,54884	65512,83438	65439,4332	65607	65650	65681	65999	66659
197	p197r5s6t7	65292,63204	65355,54727	65565,26212	65607,20469	65607	65650	65681	65999	66659
199	p199r5s6t7	65586,2332	65449,91954	65439,43321	65428,94767	65607	65650	65681	65999	66659
211	p211r5s6t7	65701,57735	65743,51955	65523,31991	65407,97657	65607	65650	65681	65999	66659
223	p223r5s6t7	65533,80547	65387,00469	65575,74845	65785,46173	65607	65650	65681	65999	66659
227	p227r5s6t7	65649,14962	65512,83399	65586,23282	65428,94728	65607	65650	65681	65999	66659
229	p229r5s6t7	65271,66095	65418,46133	65449,91874	65407,97618	65607	65650	65681	65999	66659
233	p233r5s6t7	65523,31992	65649,14844	65659,63399	65649,14843	65607	65650	65681	65999	66659
239	p239r5s6t7	65617,69219	65670,12032	65628,1762	65649,14882	65607	65650	65681	65999	66659
241	p241r5s6t7	65387,00392	65481,37617	65754,00546	65565,26211	65607	65650	65681	65999	66659
251	p251r5s6t7	65544,29102	65512,83399	65334,5754	65187,77541	65607	65650	65681	65999	66659
257	p257r4s5t7	65533,80548	65292,63242	65554,77696	65502,34845	65607	65650	65681	65999	66659
263	p263r5s6t7	65586,23437	65712,06328	65303,11875	65565,26213	65607	65650	65681	65999	66659
269	p269r5s6t7	65554,77695	65733,0344	65481,37656	65355,54649	65607	65650	65681	65999	66659
271	p271r4s5t6	65617,6914	65607,20548	65313,60431	65366,03242	65607	65650	65681	65999	66659
277	p277r5s6t7	65523,3207	65345,06173	65596,71992	65491,8625	65607	65650	65681	65999	66659
281	p281r5s6t7	65449,91876	65554,77695	65596,7207	65670,11992	65607	65650	65681	65999	66659
283	p283r5s6t7	65554,77697	65628,17735	65586,23359	65638,6629	65607	65650	65681	65999	66659
293	p293r4s6t7	65491,86248	65523,31992	65523,31875	65554,77735	65607	65650	65681	65999	66659
307	p307r5s6t7	65470,88985	65670,12071	65722,54884	65670,11994	65607	65650	65681	65999	66659
311	p311r5s6t7	65659,6336	65439,43281	65649,14844	65554,77617	65607	65650	65681	65999	66659
313	p313r5s6t7	65271,66096	65376,51876	65565,26252	65607,20508	65607	65650	65681	65999	66659

Tabla del test de póker 16 para combinaciones *min*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	0,001
3	p3r4s5t7	65680,60624	65638,66212	70587,94298	70210,45469	65607	65650	65681	65999	66659
5	p5r3s5t6	65670,11953	65607,20508	66540,43751	66603,35235	65607	65650	65681	65999	66659
7	p7r3s4t5	65439,4336	65460,4047	66016,15	65858,86406	65607	65650	65681	65999	66659
11	p11r2s4t5	65628,17735	65596,72031	65628,17698	65397,49025	65607	65650	65681	65999	66659
13	p13r2s4t5	65250,68946	65607,20585	65533,80508	65701,57694	65607	65650	65681	65999	66659
17	p17r2s4t5	65387,00429	65523,31914	65491,86289	65628,17735	65607	65650	65681	65999	66659
19	p19r2s4t5	65806,43517	65785,46446	65649,14884	65607,20432	65607	65650	65681	65999	66659
23	p23r2s3t5	65575,74844	65491,86173	65428,94844	65397,49023	65607	65650	65681	65999	66659
29	p29r2s3t5	65470,89064	65502,34767	65481,37656	65470,88985	65607	65650	65681	65999	66659
31	p31r2s3t4	65313,60469	65554,77618	65586,23244	65544,29103	65607	65650	65681	65999	66659
37	p37r2s3t4	65491,86172	65544,29062	65397,48984	65303,11837	65607	65650	65681	65999	66659
41	p41r2s3t4	65586,23321	65712,06368	65617,69102	65533,8043	65607	65650	65681	65999	66659
43	p43r2s3t4	65638,6625	65533,80508	65743,52111	65691,09104	65607	65650	65681	65999	66659
47	p47r2s3t4	65680,60547	65470,89064	65575,74884	65460,4047	65607	65650	65681	65999	66659
53	p53r2s3t4	65470,89063	65544,29141	65544,2914	65544,29025	65607	65650	65681	65999	66659
59	p59r2s3t4	65554,77696	65523,31992	65439,4332	65355,5469	65607	65650	65681	65999	66659
61	p61r2s3t4	65387,00469	65523,31876	65617,69219	65795,94805	65607	65650	65681	65999	66659
67	p67r2s3t4	65355,54767	65460,4047	65428,94766	65491,86249	65607	65650	65681	65999	66659
71	p71r2s3t4	65407,97578	65481,37695	65670,11915	65670,11876	65607	65650	65681	65999	66659
73	p73r2s3t4	65554,77695	65701,57813	65565,26329	65670,11876	65607	65650	65681	65999	66659
79	p79r2s3t4	65628,17658	65533,80586	65596,71915	65544,29062	65607	65650	65681	65999	66659
83	p83r2s3t4	65523,31875	65533,80469	65470,89024	65512,83322	65607	65650	65681	65999	66659
89	p89r2s3t4	65345,06132	65418,46135	65449,91835	65575,74688	65607	65650	65681	65999	66659
97	p97r2s3t4	65387,00431	65575,74845	65439,4336	65387,00508	65607	65650	65681	65999	66659
101	p101r2s3t4	65428,94765	65680,60664	65617,69062	65575,74807	65607	65650	65681	65999	66659
103	p103r2s3t4	65691,09142	65607,20431	65607,20587	65449,91915	65607	65650	65681	65999	66659
107	p107r1s3t4	65481,3758	65533,80546	65376,51836	65428,94767	65607	65650	65681	65999	66659
109	p109r1s3t4	65607,20508	65607,20548	65607,2051	65554,77697	65607	65650	65681	65999	66659
113	p113r1s3t4	65628,17656	65544,29181	65596,71877	65449,91914	65607	65650	65681	65999	66659
127	p127r1s3t4	65449,91837	65502,34766	65428,94766	65481,3758	65607	65650	65681	65999	66659
131	p131r1s3t4	65722,54885	65638,66213	65596,71915	65481,3758	65607	65650	65681	65999	66659
137	p137r1s3t4	65523,31876	65617,69103	65596,71993	65596,71953	65607	65650	65681	65999	66659
139	p139r1s3t4	65722,54922	65691,0914	65439,43361	65418,46212	65607	65650	65681	65999	66659
149	p149r1s3t4	65670,11953	65554,77617	65586,23439	65670,11955	65607	65650	65681	65999	66659
151	p151r1s3t4	65596,7207	65575,74726	65544,29025	65617,69141	65607	65650	65681	65999	66659
157	p157r1s3t4	65449,91837	65533,80509	65502,34805	65366,0332	65607	65650	65681	65999	66659
163	p163r1s3t4	65607,20509	65512,8336	65586,23361	65439,43282	65607	65650	65681	65999	66659
167	p167r1s3t4	65617,69025	65670,11994	65533,80509	65523,31915	65607	65650	65681	65999	66659
173	p173r1s3t4	65565,26134	65628,17581	65387,00391	65292,63282	65607	65650	65681	65999	66659
179	p179r1s3t4	65596,71915	65774,97697	65544,29063	65334,57618	65607	65650	65681	65999	66659
181	p181r1s3t4	65670,11993	65659,63397	65523,31875	65837,89298	65607	65650	65681	65999	66659
191	p191r1s3t4	65428,94727	65491,86289	65512,83361	65512,83359	65607	65650	65681	65999	66659
193	p193r1s3t4	65659,63438	65712,06368	65481,37618	65376,51837	65607	65650	65681	65999	66659
197	p197r1s3t4	65324,09025	65586,23439	65292,63282	65240,20313	65607	65650	65681	65999	66659
199	p199r1s3t4	65554,77697	65754,00626	65607,20509	65523,31915	65607	65650	65681	65999	66659
211	p211r1s3t4	65575,74844	65439,43283	65502,34728	65533,80508	65607	65650	65681	65999	66659
223	p223r1s3t4	65638,66328	65554,77618	65575,74804	65701,57774	65607	65650	65681	65999	66659
227	p227r1s3t4	65554,77735	65691,0926	65544,29024	65523,31876	65607	65650	65681	65999	66659
229	p229r1s3t4	65649,14803	65376,51916	65533,80508	65670,11991	65607	65650	65681	65999	66659
233	p233r1s3t4	65502,34728	65397,49063	65376,51876	65491,86136	65607	65650	65681	65999	66659
239	p239r1s3t4	65575,74884	65712,06172	65565,26328	65481,37657	65607	65650	65681	65999	66659
241	p241r1s3t4	65628,17813	65785,46485	65512,8344	65460,40432	65607	65650	65681	65999	66659
251	p251r1s3t4	65638,6629	65554,77658	65628,17697	65743,5203	65607	65650	65681	65999	66659
257	p257r1s3t4	65764,49219	65617,69101	65617,69142	65460,40471	65607	65650	65681	65999	66659
263	p263r1s3t4	65366,03205	65470,88946	65628,17654	65596,71993	65607	65650	65681	65999	66659
269	p269r1s3t4	65481,37656	65439,43243	65460,40471	65303,11837	65607	65650	65681	65999	66659
271	p271r1s3t4	65533,80472	65491,86212	65586,23477	65680,60548	65607	65650	65681	65999	66659
277	p277r1s3t4	65407,97579	65460,40508	65418,46094	65418,46133	65607	65650	65681	65999	66659
281	p281r1s3t4	65837,89298	65722,54805	65334,57541	65355,54728	65607	65650	65681	65999	66659
283	p283r1s3t4	65544,28985	65900,80666	65586,23399	65586,23439	65607	65650	65681	65999	66659
293	p293r1s3t4	65439,43322	65575,74727	65428,94728	65460,4043	65607	65650	65681	65999	66659
307	p307r1s3t4	65649,14884	65701,57774	65502,34727	65554,77578	65607	65650	65681	65999	66659
311	p311r1s3t4	65449,91954	65586,23398	65722,54883	65774,97694	65607	65650	65681	65999	66659
313	p313r1s3t4	65481,37657	65523,31875	65397,49023	65470,89062	65607	65650	65681	65999	66659

Tabla del test de póker 16 para combinaciones *bottom*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	0,001
3	p3r4s5t7	65680,60624	65638,66212	70587,94298	70210,45469	65607	65650	65681	65999	66659
5	p5r2s5t7	65397,48984	65366,03245	66802,58205	66425,09452	65607	65650	65681	65999	66659
7	p7r1s5t7	65355,54688	65502,34766	66152,46523	65974,20782	65607	65650	65681	65999	66659
11	p11r1s4t7	65565,26251	65449,91993	65596,71837	65638,66211	65607	65650	65681	65999	66659
13	p13r1s5t6	65554,77694	65575,74805	65649,14922	65659,63399	65607	65650	65681	65999	66659
17	p17r1s4t6	65502,34688	65806,43517	65607,20664	65565,26134	65607	65650	65681	65999	66659
19	p19r1s3t7	65628,17736	65596,71877	65512,8336	65491,86212	65607	65650	65681	65999	66659
23	p23r1s3t7	65554,77734	65481,37618	65449,91837	65460,4047	65607	65650	65681	65999	66659
29	p29r1s3t7	65680,60625	65680,60625	65481,37658	65596,72031	65607	65650	65681	65999	66659
31	p31r1s3t7	65544,29141	65523,31877	65512,83321	65575,74766	65607	65650	65681	65999	66659
37	p37r1s3t7	65575,74728	65628,17736	65345,06097	65512,83398	65607	65650	65681	65999	66659
41	p41r1s3t7	65523,318	65691,09218	65502,34844	65397,49063	65607	65650	65681	65999	66659
43	p43r1s3t7	65523,31837	65460,40471	65324,08984	65303,11835	65607	65650	65681	65999	66659
47	p47r1s3t7	65900,8078	65712,06328	65407,97501	65376,51876	65607	65650	65681	65999	66659
53	p53r1s2t7	65659,63516	65659,63478	65366,03283	65313,60432	65607	65650	65681	65999	66659
59	p59r1s3t5	65586,23438	65502,34765	65439,43321	65491,86211	65607	65650	65681	65999	66659
61	p61r1s2t7	65544,29104	65428,94806	65628,17657	65586,234	65607	65650	65681	65999	66659
67	p67r1s3t5	65481,37695	65565,26212	65617,69141	65596,72032	65607	65650	65681	65999	66659
71	p71r1s3t5	65596,71914	65628,17658	65407,97539	65449,91954	65607	65650	65681	65999	66659
73	p73r1s2t7	65502,34727	65512,83398	65596,71955	65523,31835	65607	65650	65681	65999	66659
79	p79r1s3t5	65607,20627	65439,43399	65554,77696	65554,77697	65607	65650	65681	65999	66659
83	p83r1s3t5	65481,37579	65554,77736	65670,1207	65544,29142	65607	65650	65681	65999	66659
89	p89r1s3t5	65481,37617	65481,37736	65544,29142	65701,57851	65607	65650	65681	65999	66659
97	p97r1s2t7	65596,71993	65659,63437	65533,80509	65628,17619	65607	65650	65681	65999	66659
101	p101r1s3t5	65565,2625	65544,29101	65607,20508	65743,52071	65607	65650	65681	65999	66659
103	p103r1s2t7	65512,8336	65418,46095	65324,08946	65334,57539	65607	65650	65681	65999	66659
107	p107r1s3t4	65481,3758	65533,80546	65376,51836	65428,94767	65607	65650	65681	65999	66659
109	p109r1s3t4	65607,20508	65607,20548	65607,2051	65554,77697	65607	65650	65681	65999	66659
113	p113r1s3t4	65628,17656	65544,29181	65596,71877	65449,91914	65607	65650	65681	65999	66659
127	p127r1s3t4	65449,91837	65502,34766	65428,94766	65481,3758	65607	65650	65681	65999	66659
131	p131r1s3t4	65722,54885	65638,66213	65596,71915	65481,3758	65607	65650	65681	65999	66659
137	p137r1s2t7	65586,23476	65428,9473	65407,97541	65303,11799	65607	65650	65681	65999	66659
139	p139r1s3t4	65722,54922	65691,0914	65439,43361	65418,46212	65607	65650	65681	65999	66659
149	p149r1s3t4	65670,11953	65554,77617	65586,23439	65670,11955	65607	65650	65681	65999	66659
151	p151r1s2t7	65586,23398	65586,2336	65816,91991	65638,66329	65607	65650	65681	65999	66659
157	p157r1s2t7	65733,03476	65638,66212	65596,71993	65533,80469	65607	65650	65681	65999	66659
163	p163r1s2t7	65544,29101	65586,23399	65816,92187	65953,23595	65607	65650	65681	65999	66659
167	p167r1s3t4	65617,69025	65670,11994	65533,80509	65523,31915	65607	65650	65681	65999	66659
173	p173r1s3t4	65565,26134	65628,17581	65387,00391	65292,63282	65607	65650	65681	65999	66659
179	p179r1s3t4	65596,71915	65774,97697	65544,29063	65334,57618	65607	65650	65681	65999	66659
181	p181r1s3t4	65670,11993	65659,63397	65523,31875	65837,89298	65607	65650	65681	65999	66659
191	p191r1s3t4	65428,94727	65491,86289	65512,83361	65512,83359	65607	65650	65681	65999	66659
193	p193r1s2t5	65387,0047	65523,31953	65366,0332	65292,63281	65607	65650	65681	65999	66659
197	p197r1s3t4	65324,09025	65586,23439	65292,63282	65240,20313	65607	65650	65681	65999	66659
199	p199r1s3t4	65554,77697	65754,00626	65607,20509	65523,31915	65607	65650	65681	65999	66659
211	p211r1s2t5	65303,11837	65240,20391	65470,89063	65544,28984	65607	65650	65681	65999	66659
223	p223r1s3t4	65638,66328	65554,77618	65575,74804	65701,57774	65607	65650	65681	65999	66659
227	p227r1s3t4	65554,77735	65691,0926	65544,29024	65523,31876	65607	65650	65681	65999	66659
229	p229r1s2t5	65502,34766	65481,37541	65428,94687	65407,9754	65607	65650	65681	65999	66659
233	p233r1s3t4	65502,34728	65397,49063	65376,51876	65491,86136	65607	65650	65681	65999	66659
239	p239r1s3t4	65575,74884	65712,06172	65565,26328	65481,37657	65607	65650	65681	65999	66659
241	p241r1s2t5	65649,14844	65554,77578	65397,49023	65470,89102	65607	65650	65681	65999	66659
251	p251r1s3t4	65638,6629	65554,77658	65628,17697	65743,5203	65607	65650	65681	65999	66659
257	p257r1s2t5	65533,80431	65324,09025	65544,29063	65680,60586	65607	65650	65681	65999	66659
263	p263r1s2t5	65481,37618	65659,63437	65533,8043	65460,40353	65607	65650	65681	65999	66659
269	p269r1s3t4	65481,37656	65439,43243	65460,40471	65303,11837	65607	65650	65681	65999	66659
271	p271r1s2t5	65523,31915	65575,74806	65292,63282	65439,43361	65607	65650	65681	65999	66659
277	p277r1s2t5	65565,2633	65544,29023	65554,77696	65481,37618	65607	65650	65681	65999	66659
281	p281r1s2t5	65366,03242	65397,48946	65617,6914	65649,14923	65607	65650	65681	65999	66659
283	p283r1s2t5	65544,29024	65607,20625	65502,34728	65607,20507	65607	65650	65681	65999	66659
293	p293r1s2t5	65554,77695	65491,86172	65596,71916	65449,91876	65607	65650	65681	65999	66659
307	p307r1s3t4	65649,14884	65701,57774	65502,34727	65554,77578	65607	65650	65681	65999	66659
311	p311r1s3t4	65449,91954	65586,23398	65722,54883	65774,97694	65607	65650	65681	65999	66659
313	p313r1s2t5	65649,14844	65439,43281	65659,6336	65502,34688	65607	65650	65681	65999	66659

Tabla del test de rachas para combinaciones *top*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	0,001
3	p3r5s6t7	17,04795	16,09847	2074,16658	2782,48662	16,1032	15,9688	15,7268	23,5418	39,2524
5	p5r5s6t7	17,87063	19,7932	742,11665	1082,1592	16,1032	15,9688	15,7268	23,5418	39,2524
7	p7r5s6t7	17,03883	17,18363	396,45767	591,64288	16,1032	15,9688	15,7268	23,5418	39,2524
11	p11r5s6t7	15,95	15,3251	180,01309	259,10575	16,1032	15,9688	15,7268	23,5418	39,2524
13	p13r5s6t7	17,41074	18,25066	132,8783	202,28617	16,1032	15,9688	15,7268	23,5418	39,2524
17	p17r5s6t7	18,59989	18,98022	75,99414	111,71917	16,1032	15,9688	15,7268	23,5418	39,2524
19	p19r5s6t7	14,5882	15,77251	75,30915	110,94398	16,1032	15,9688	15,7268	23,5418	39,2524
23	p23r5s6t7	12,77209	14,39253	51,97877	73,02274	16,1032	15,9688	15,7268	23,5418	39,2524
29	p29r5s6t7	16,4426	15,93604	42,04827	55,3979	16,1032	15,9688	15,7268	23,5418	39,2524
31	p31r5s6t7	14,861	15,82861	40,21293	56,08455	16,1032	15,9688	15,7268	23,5418	39,2524
37	p37r5s6t7	17,44063	18,78875	26,55729	34,32818	16,1032	15,9688	15,7268	23,5418	39,2524
41	p41r5s6t7	14,99477	13,48482	24,1081	27,90708	16,1032	15,9688	15,7268	23,5418	39,2524
43	p43r5s6t7	16,74521	15,65628	33,71192	38,17351	16,1032	15,9688	15,7268	23,5418	39,2524
47	p47r5s6t7	12,67456	14,49633	25,81576	30,57469	16,1032	15,9688	15,7268	23,5418	39,2524
53	p53r5s6t7	20,07514	17,50745	22,44999	26,74296	16,1032	15,9688	15,7268	23,5418	39,2524
59	p59r5s6t7	16,92079	15,22818	20,30905	24,49032	16,1032	15,9688	15,7268	23,5418	39,2524
61	p61r5s6t7	14,31439	18,15921	21,58361	25,08082	16,1032	15,9688	15,7268	23,5418	39,2524
67	p67r5s6t7	15,83016	14,18777	19,8618	22,56402	16,1032	15,9688	15,7268	23,5418	39,2524
71	p71r5s6t7	14,43448	15,23263	21,27933	21,02487	16,1032	15,9688	15,7268	23,5418	39,2524
73	p73r5s6t7	17,39604	19,94498	20,56183	21,68454	16,1032	15,9688	15,7268	23,5418	39,2524
79	p79r5s6t7	17,0036	14,82034	21,50972	26,5749	16,1032	15,9688	15,7268	23,5418	39,2524
83	p83r5s6t7	18,46805	17,94119	17,50463	16,52397	16,1032	15,9688	15,7268	23,5418	39,2524
89	p89r5s6t7	16,42812	19,57848	21,04786	19,47448	16,1032	15,9688	15,7268	23,5418	39,2524
97	p97r5s6t7	16,52655	15,14933	16,77476	18,48886	16,1032	15,9688	15,7268	23,5418	39,2524
101	p101r5s6t7	15,18032	15,83587	19,31876	18,5949	16,1032	15,9688	15,7268	23,5418	39,2524
103	p103r5s6t7	18,63524	16,83495	15,57914	16,88859	16,1032	15,9688	15,7268	23,5418	39,2524
107	p107r5s6t7	14,77982	18,68316	19,5029	17,11046	16,1032	15,9688	15,7268	23,5418	39,2524
109	p109r5s6t7	15,8995	15,06706	21,48765	19,84242	16,1032	15,9688	15,7268	23,5418	39,2524
113	p113r5s6t7	16,66785	16,43785	18,64187	20,60833	16,1032	15,9688	15,7268	23,5418	39,2524
127	p127r5s6t7	16,65063	16,85631	15,62525	15,9682	16,1032	15,9688	15,7268	23,5418	39,2524
131	p131r5s6t7	14,72724	17,31529	19,11344	22,33897	16,1032	15,9688	15,7268	23,5418	39,2524
137	p137r5s6t7	16,84904	18,09131	18,57184	18,43244	16,1032	15,9688	15,7268	23,5418	39,2524
139	p139r5s6t7	18,32632	15,55314	16,30672	19,43186	16,1032	15,9688	15,7268	23,5418	39,2524
149	p149r5s6t7	16,25161	16,44877	15,91924	16,68718	16,1032	15,9688	15,7268	23,5418	39,2524
151	p151r5s6t7	16,23238	17,29819	15,81317	17,39146	16,1032	15,9688	15,7268	23,5418	39,2524
157	p157r5s6t7	19,82611	18,57105	20,33514	19,47704	16,1032	15,9688	15,7268	23,5418	39,2524
163	p163r5s6t7	16,17358	16,95142	15,74753	19,41309	16,1032	15,9688	15,7268	23,5418	39,2524
167	p167r5s6t7	14,21652	15,01467	14,91388	18,54758	16,1032	15,9688	15,7268	23,5418	39,2524
173	p173r4s5t6	14,96342	14,60809	20,95684	20,80849	16,1032	15,9688	15,7268	23,5418	39,2524
179	p179r5s6t7	15,38236	16,4059	17,74157	16,77951	16,1032	15,9688	15,7268	23,5418	39,2524
181	p181r5s6t7	15,25194	14,24122	17,09817	16,0741	16,1032	15,9688	15,7268	23,5418	39,2524
191	p191r5s6t7	13,58323	14,17637	18,62753	19,85462	16,1032	15,9688	15,7268	23,5418	39,2524
193	p193r5s6t7	16,12215	15,22927	17,16464	16,18702	16,1032	15,9688	15,7268	23,5418	39,2524
197	p197r5s6t7	18,64484	17,67812	17,22206	15,1582	16,1032	15,9688	15,7268	23,5418	39,2524
199	p199r5s6t7	17,12767	18,33223	18,36854	17,41602	16,1032	15,9688	15,7268	23,5418	39,2524
211	p211r5s6t7	16,14751	17,32886	17,60134	16,38773	16,1032	15,9688	15,7268	23,5418	39,2524
223	p223r5s6t7	14,48229	16,1626	18,87999	20,325	16,1032	15,9688	15,7268	23,5418	39,2524
227	p227r5s6t7	16,43445	16,71739	16,39638	17,46935	16,1032	15,9688	15,7268	23,5418	39,2524
229	p229r5s6t7	13,94992	14,19341	15,95615	14,35425	16,1032	15,9688	15,7268	23,5418	39,2524
233	p233r5s6t7	15,54557	14,3569	16,71913	17,47672	16,1032	15,9688	15,7268	23,5418	39,2524
239	p239r5s6t7	14,26061	17,28657	15,05702	17,25359	16,1032	15,9688	15,7268	23,5418	39,2524
241	p241r5s6t7	19,39354	20,94157	19,61616	20,58185	16,1032	15,9688	15,7268	23,5418	39,2524
251	p251r5s6t7	16,00965	14,02893	13,86998	17,06395	16,1032	15,9688	15,7268	23,5418	39,2524
257	p257r4s5t7	16,85799	16,56845	20,82587	18,09292	16,1032	15,9688	15,7268	23,5418	39,2524
263	p263r5s6t7	16,0325	17,49816	17,25714	17,03507	16,1032	15,9688	15,7268	23,5418	39,2524
269	p269r5s6t7	19,3349	16,55273	17,55352	17,42649	16,1032	15,9688	15,7268	23,5418	39,2524
271	p271r4s5t6	20,10658	19,94385	17,86533	15,98751	16,1032	15,9688	15,7268	23,5418	39,2524
277	p277r5s6t7	15,02935	15,80093	19,1397	19,41679	16,1032	15,9688	15,7268	23,5418	39,2524
281	p281r5s6t7	19,43922	18,61112	14,84593	15,53928	16,1032	15,9688	15,7268	23,5418	39,2524
283	p283r5s6t7	14,47729	16,27918	17,92136	16,81422	16,1032	15,9688	15,7268	23,5418	39,2524
293	p293r4s6t7	14,60364	17,82907	14,33284	16,95285	16,1032	15,9688	15,7268	23,5418	39,2524
307	p307r5s6t7	17,80427	16,04377	15,51457	18,19113	16,1032	15,9688	15,7268	23,5418	39,2524
311	p311r5s6t7	13,93146	17,75278	14,41699	14,26949	16,1032	15,9688	15,7268	23,5418	39,2524
313	p313r5s6t7	15,02294	17,14974	16,67291	15,74754	16,1032	15,9688	15,7268	23,5418	39,2524

Tabla del test de rachas para combinaciones *min*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	15,41388	12,48655	2079,12664	2769,72182	16,1032	15,9688	15,7268	23,5418	39,2524
5	p5r3s5t6	15,71655	18,79049	796,67313	1113,91209	16,1032	15,9688	15,7268	23,5418	39,2524
7	p7r3s4t5	16,40125	15,84599	414,33006	600,87336	16,1032	15,9688	15,7268	23,5418	39,2524
11	p11r2s4t5	15,4503	14,36787	161,68316	237,0216	16,1032	15,9688	15,7268	23,5418	39,2524
13	p13r2s4t5	15,38364	15,03283	136,63068	200,45577	16,1032	15,9688	15,7268	23,5418	39,2524
17	p17r2s4t5	15,54135	16,97192	82,68833	113,35684	16,1032	15,9688	15,7268	23,5418	39,2524
19	p19r2s4t5	12,95189	12,16547	61,08292	89,9629	16,1032	15,9688	15,7268	23,5418	39,2524
23	p23r2s3t5	16,56599	16,35004	53,49112	76,13015	16,1032	15,9688	15,7268	23,5418	39,2524
29	p29r2s3t5	19,37994	15,95663	39,83431	46,4391	16,1032	15,9688	15,7268	23,5418	39,2524
31	p31r2s3t4	15,88659	15,03669	40,54293	51,15554	16,1032	15,9688	15,7268	23,5418	39,2524
37	p37r2s3t4	18,68376	16,98571	30,59266	42,72648	16,1032	15,9688	15,7268	23,5418	39,2524
41	p41r2s3t4	16,29795	16,46652	24,9669	30,14078	16,1032	15,9688	15,7268	23,5418	39,2524
43	p43r2s3t4	16,89819	14,24962	25,00132	33,22656	16,1032	15,9688	15,7268	23,5418	39,2524
47	p47r2s3t4	17,78814	17,56547	22,86828	26,03593	16,1032	15,9688	15,7268	23,5418	39,2524
53	p53r2s3t4	15,00757	12,69066	18,087	23,51438	16,1032	15,9688	15,7268	23,5418	39,2524
59	p59r2s3t4	16,35811	17,845	24,07523	30,41606	16,1032	15,9688	15,7268	23,5418	39,2524
61	p61r2s3t4	15,36956	12,88953	19,12892	22,57654	16,1032	15,9688	15,7268	23,5418	39,2524
67	p67r2s3t4	15,68621	19,39011	21,68651	24,1715	16,1032	15,9688	15,7268	23,5418	39,2524
71	p71r2s3t4	18,60884	20,00023	18,23983	21,45801	16,1032	15,9688	15,7268	23,5418	39,2524
73	p73r2s3t4	17,88044	17,53562	22,40604	20,43895	16,1032	15,9688	15,7268	23,5418	39,2524
79	p79r2s3t4	17,7211	16,63773	20,64626	22,22652	16,1032	15,9688	15,7268	23,5418	39,2524
83	p83r2s3t4	14,41441	16,68576	23,29131	22,93015	16,1032	15,9688	15,7268	23,5418	39,2524
89	p89r2s3t4	18,30095	15,44092	17,48321	15,50291	16,1032	15,9688	15,7268	23,5418	39,2524
97	p97r2s3t4	17,17678	17,20541	22,7502	24,13086	16,1032	15,9688	15,7268	23,5418	39,2524
101	p101r2s3t4	15,15896	16,91817	16,17249	17,0253	16,1032	15,9688	15,7268	23,5418	39,2524
103	p103r2s3t4	13,50964	18,26705	20,01605	21,01163	16,1032	15,9688	15,7268	23,5418	39,2524
107	p107r1s3t4	16,5677	17,63419	17,07798	15,68291	16,1032	15,9688	15,7268	23,5418	39,2524
109	p109r1s3t4	19,79005	19,73576	19,57145	18,8928	16,1032	15,9688	15,7268	23,5418	39,2524
113	p113r1s3t4	19,08588	16,82543	17,23825	19,25044	16,1032	15,9688	15,7268	23,5418	39,2524
127	p127r1s3t4	18,99144	16,45757	17,39328	21,62184	16,1032	15,9688	15,7268	23,5418	39,2524
131	p131r1s3t4	17,23718	19,03047	15,14215	18,30564	16,1032	15,9688	15,7268	23,5418	39,2524
137	p137r1s3t4	19,10501	19,19902	14,82251	18,59158	16,1032	15,9688	15,7268	23,5418	39,2524
139	p139r1s3t4	17,06491	16,20042	17,64907	16,98383	16,1032	15,9688	15,7268	23,5418	39,2524
149	p149r1s3t4	17,14864	19,29252	16,58783	18,53552	16,1032	15,9688	15,7268	23,5418	39,2524
151	p151r1s3t4	16,59721	15,17192	20,95866	24,05728	16,1032	15,9688	15,7268	23,5418	39,2524
157	p157r1s3t4	19,47082	19,50825	14,82934	12,55339	16,1032	15,9688	15,7268	23,5418	39,2524
163	p163r1s3t4	18,31618	16,70203	18,44198	17,37321	16,1032	15,9688	15,7268	23,5418	39,2524
167	p167r1s3t4	14,63459	18,47586	14,74512	15,93864	16,1032	15,9688	15,7268	23,5418	39,2524
173	p173r1s3t4	18,06885	18,78584	15,43989	16,48629	16,1032	15,9688	15,7268	23,5418	39,2524
179	p179r1s3t4	15,60025	19,58128	18,88428	18,96382	16,1032	15,9688	15,7268	23,5418	39,2524
181	p181r1s3t4	17,60938	17,86614	17,96296	18,99871	16,1032	15,9688	15,7268	23,5418	39,2524
191	p191r1s3t4	19,19505	17,01613	19,61506	17,27672	16,1032	15,9688	15,7268	23,5418	39,2524
193	p193r1s3t4	17,65696	17,59655	13,96754	13,4224	16,1032	15,9688	15,7268	23,5418	39,2524
197	p197r1s3t4	16,69794	15,8025	19,8946	22,51476	16,1032	15,9688	15,7268	23,5418	39,2524
199	p199r1s3t4	18,00904	18,78495	15,99524	15,90551	16,1032	15,9688	15,7268	23,5418	39,2524
211	p211r1s3t4	15,54306	16,75008	13,00794	18,92411	16,1032	15,9688	15,7268	23,5418	39,2524
223	p223r1s3t4	14,9011	14,56178	13,45651	14,45351	16,1032	15,9688	15,7268	23,5418	39,2524
227	p227r1s3t4	16,15965	15,82126	17,25559	14,44356	16,1032	15,9688	15,7268	23,5418	39,2524
229	p229r1s3t4	16,11306	18,23304	16,00034	15,0316	16,1032	15,9688	15,7268	23,5418	39,2524
233	p233r1s3t4	16,29242	15,53478	15,45445	15,29671	16,1032	15,9688	15,7268	23,5418	39,2524
239	p239r1s3t4	18,02218	17,00198	18,04988	16,32581	16,1032	15,9688	15,7268	23,5418	39,2524
241	p241r1s3t4	18,57984	16,90673	17,50159	16,02438	16,1032	15,9688	15,7268	23,5418	39,2524
251	p251r1s3t4	18,36872	15,89684	16,23185	16,10954	16,1032	15,9688	15,7268	23,5418	39,2524
257	p257r1s3t4	15,38471	13,89637	17,88105	19,09688	16,1032	15,9688	15,7268	23,5418	39,2524
263	p263r1s3t4	18,01269	15,13312	15,17403	15,58744	16,1032	15,9688	15,7268	23,5418	39,2524
269	p269r1s3t4	15,74473	14,35551	15,65078	15,18192	16,1032	15,9688	15,7268	23,5418	39,2524
271	p271r1s3t4	14,82428	16,28924	17,52163	22,01	16,1032	15,9688	15,7268	23,5418	39,2524
277	p277r1s3t4	19,65504	21,07963	15,17069	13,87396	16,1032	15,9688	15,7268	23,5418	39,2524
281	p281r1s3t4	13,19617	13,41746	16,33702	16,28555	16,1032	15,9688	15,7268	23,5418	39,2524
283	p283r1s3t4	15,31733	13,3304	19,48423	20,98653	16,1032	15,9688	15,7268	23,5418	39,2524
293	p293r1s3t4	18,77271	15,86248	17,30286	16,04414	16,1032	15,9688	15,7268	23,5418	39,2524
307	p307r1s3t4	16,59894	16,79283	15,15905	13,57231	16,1032	15,9688	15,7268	23,5418	39,2524
311	p311r1s3t4	13,17306	13,71165	17,07796	16,39175	16,1032	15,9688	15,7268	23,5418	39,2524
313	p313r1s3t4	13,78588	15,15437	13,66843	19,02309	16,1032	15,9688	15,7268	23,5418	39,2524

Tabla del test de rachas para combinaciones *bottom*

Pr.	Comb.	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	0,001
3	p3r4s5t7	15,41388	12,48655	2079,12664	2769,72182	16,1032	15,9688	15,7268	23,5418	39,2524
5	p5r2s5t7	17,08912	15,92636	759,51637	1081,98263	16,1032	15,9688	15,7268	23,5418	39,2524
7	p7r1s5t7	15,80308	15,48719	416,70081	610,20317	16,1032	15,9688	15,7268	23,5418	39,2524
11	p11r1s4t7	16,10264	14,67547	166,53396	244,36411	16,1032	15,9688	15,7268	23,5418	39,2524
13	p13r1s5t6	15,10057	15,46295	110,99685	163,16594	16,1032	15,9688	15,7268	23,5418	39,2524
17	p17r1s4t6	18,21634	14,23165	75,37562	111,86481	16,1032	15,9688	15,7268	23,5418	39,2524
19	p19r1s3t7	16,80043	14,27823	63,45947	92,35495	16,1032	15,9688	15,7268	23,5418	39,2524
23	p23r1s3t7	16,81937	18,62575	50,34842	74,00838	16,1032	15,9688	15,7268	23,5418	39,2524
29	p29r1s3t7	19,15284	19,82892	39,25429	51,14735	16,1032	15,9688	15,7268	23,5418	39,2524
31	p31r1s3t7	17,77909	18,34964	34,53239	43,59846	16,1032	15,9688	15,7268	23,5418	39,2524
37	p37r1s3t7	17,2981	14,75447	34,8049	42,53108	16,1032	15,9688	15,7268	23,5418	39,2524
41	p41r1s3t7	16,28421	16,23396	26,40436	31,12564	16,1032	15,9688	15,7268	23,5418	39,2524
43	p43r1s3t7	14,45662	15,24017	35,57069	47,81918	16,1032	15,9688	15,7268	23,5418	39,2524
47	p47r1s3t7	16,39202	19,57543	26,94222	32,4669	16,1032	15,9688	15,7268	23,5418	39,2524
53	p53r1s2t7	18,1701	17,97993	21,23568	26,51215	16,1032	15,9688	15,7268	23,5418	39,2524
59	p59r1s3t5	14,20447	17,25284	16,96996	21,91213	16,1032	15,9688	15,7268	23,5418	39,2524
61	p61r1s2t7	16,27502	16,3443	25,24979	30,23782	16,1032	15,9688	15,7268	23,5418	39,2524
67	p67r1s3t5	13,97733	15,02203	21,67869	24,30291	16,1032	15,9688	15,7268	23,5418	39,2524
71	p71r1s3t5	14,62879	15,77381	13,6392	14,79923	16,1032	15,9688	15,7268	23,5418	39,2524
73	p73r1s2t7	17,4006	19,20576	19,34988	20,54398	16,1032	15,9688	15,7268	23,5418	39,2524
79	p79r1s3t5	16,81816	17,27322	19,52231	21,05628	16,1032	15,9688	15,7268	23,5418	39,2524
83	p83r1s3t5	15,17026	16,60062	19,41303	20,03927	16,1032	15,9688	15,7268	23,5418	39,2524
89	p89r1s3t5	16,23054	16,75031	20,35974	18,844	16,1032	15,9688	15,7268	23,5418	39,2524
97	p97r1s2t7	16,25188	17,40418	15,40026	16,16361	16,1032	15,9688	15,7268	23,5418	39,2524
101	p101r1s3t5	17,40701	16,86735	19,33963	19,17698	16,1032	15,9688	15,7268	23,5418	39,2524
103	p103r1s2t7	17,73848	16,96861	14,79335	14,27013	16,1032	15,9688	15,7268	23,5418	39,2524
107	p107r1s3t4	16,5677	17,63419	17,07798	15,68291	16,1032	15,9688	15,7268	23,5418	39,2524
109	p109r1s3t4	19,79005	19,73576	19,57145	18,8928	16,1032	15,9688	15,7268	23,5418	39,2524
113	p113r1s3t4	19,08588	16,82543	17,23825	19,25044	16,1032	15,9688	15,7268	23,5418	39,2524
127	p127r1s3t4	18,99144	16,45757	17,39328	21,62184	16,1032	15,9688	15,7268	23,5418	39,2524
131	p131r1s3t4	17,23718	19,03047	15,14215	18,30564	16,1032	15,9688	15,7268	23,5418	39,2524
137	p137r1s2t7	15,22097	14,67827	15,56592	15,75355	16,1032	15,9688	15,7268	23,5418	39,2524
139	p139r1s3t4	17,06491	16,20042	17,64907	16,98383	16,1032	15,9688	15,7268	23,5418	39,2524
149	p149r1s3t4	17,14864	19,29252	16,58783	18,53552	16,1032	15,9688	15,7268	23,5418	39,2524
151	p151r1s2t7	18,27143	15,96006	14,75624	19,78566	16,1032	15,9688	15,7268	23,5418	39,2524
157	p157r1s2t7	12,18388	12,81775	19,79616	21,72342	16,1032	15,9688	15,7268	23,5418	39,2524
163	p163r1s2t7	14,62422	15,70933	15,85534	14,93503	16,1032	15,9688	15,7268	23,5418	39,2524
167	p167r1s3t4	14,63459	18,47586	14,74512	15,93864	16,1032	15,9688	15,7268	23,5418	39,2524
173	p173r1s3t4	18,06885	18,78584	15,43989	16,48629	16,1032	15,9688	15,7268	23,5418	39,2524
179	p179r1s3t4	15,60025	19,58128	18,88428	18,96382	16,1032	15,9688	15,7268	23,5418	39,2524
181	p181r1s3t4	17,60938	17,86614	17,96296	18,99871	16,1032	15,9688	15,7268	23,5418	39,2524
191	p191r1s3t4	19,19505	17,01613	19,61506	17,27672	16,1032	15,9688	15,7268	23,5418	39,2524
193	p193r1s2t5	13,78062	13,72637	14,24151	15,2598	16,1032	15,9688	15,7268	23,5418	39,2524
197	p197r1s3t4	16,69794	15,8025	19,8946	22,51476	16,1032	15,9688	15,7268	23,5418	39,2524
199	p199r1s3t4	18,00904	18,78495	15,99524	15,90551	16,1032	15,9688	15,7268	23,5418	39,2524
211	p211r1s2t5	19,58979	18,82839	18,18294	18,69216	16,1032	15,9688	15,7268	23,5418	39,2524
223	p223r1s3t4	14,9011	14,56178	13,45651	14,45351	16,1032	15,9688	15,7268	23,5418	39,2524
227	p227r1s3t4	16,15965	15,82126	17,25559	14,44356	16,1032	15,9688	15,7268	23,5418	39,2524
229	p229r1s2t5	15,06596	14,99437	16,74474	17,17651	16,1032	15,9688	15,7268	23,5418	39,2524
233	p233r1s3t4	16,29242	15,53478	15,45445	15,29671	16,1032	15,9688	15,7268	23,5418	39,2524
239	p239r1s3t4	18,02218	17,00198	18,04988	16,32581	16,1032	15,9688	15,7268	23,5418	39,2524
241	p241r1s2t5	14,83326	19,43784	17,51811	15,16105	16,1032	15,9688	15,7268	23,5418	39,2524
251	p251r1s3t4	18,36872	15,89684	16,23185	16,10954	16,1032	15,9688	15,7268	23,5418	39,2524
257	p257r1s2t5	12,2494	13,41426	15,39977	16,04346	16,1032	15,9688	15,7268	23,5418	39,2524
263	p263r1s2t5	15,69368	16,52903	16,16262	16,07691	16,1032	15,9688	15,7268	23,5418	39,2524
269	p269r1s3t4	15,74473	14,35551	15,65078	15,18192	16,1032	15,9688	15,7268	23,5418	39,2524
271	p271r1s2t5	19,33419	16,95316	17,90154	17,75778	16,1032	15,9688	15,7268	23,5418	39,2524
277	p277r1s2t5	14,37145	15,22334	17,26006	15,86219	16,1032	15,9688	15,7268	23,5418	39,2524
281	p281r1s2t5	19,59095	18,23829	17,64493	15,95256	16,1032	15,9688	15,7268	23,5418	39,2524
283	p283r1s2t5	16,37769	14,85691	16,04203	17,4473	16,1032	15,9688	15,7268	23,5418	39,2524
293	p293r1s2t5	20,75651	17,42002	13,00415	14,96944	16,1032	15,9688	15,7268	23,5418	39,2524
307	p307r1s3t4	16,59894	16,79283	15,15905	13,57231	16,1032	15,9688	15,7268	23,5418	39,2524
311	p311r1s3t4	13,17306	13,71165	17,07796	16,39175	16,1032	15,9688	15,7268	23,5418	39,2524
313	p313r1s2t5	17,05877	15,73133	15,99155	20,51023	16,1032	15,9688	15,7268	23,5418	39,2524

Tabla del test de autocorrelación para combinaciones *top*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r5s6t7	0,80025	0,80093	13,24312	0,89311	0,7978	0,7984	0,7972	1,282	3,09
5	p5r5s6t7	0,79792	0,79944	4,79083	0,83547	0,7978	0,7984	0,7972	1,282	3,09
7	p7r5s6t7	0,79475	0,79957	2,48288	0,8168	0,7978	0,7984	0,7972	1,282	3,09
11	p11r5s6t7	0,79251	0,80092	1,21558	0,80485	0,7978	0,7984	0,7972	1,282	3,09
13	p13r5s6t7	0,80006	0,79796	1,0311	0,80596	0,7978	0,7984	0,7972	1,282	3,09
17	p17r5s6t7	0,79799	0,79714	0,86775	0,80123	0,7978	0,7984	0,7972	1,282	3,09
19	p19r5s6t7	0,7968	0,80415	0,86285	0,80546	0,7978	0,7984	0,7972	1,282	3,09
23	p23r5s6t7	0,79284	0,79827	0,81968	0,79932	0,7978	0,7984	0,7972	1,282	3,09
29	p29r5s6t7	0,80066	0,79852	0,81064	0,79511	0,7978	0,7984	0,7972	1,282	3,09
31	p31r5s6t7	0,79649	0,79989	0,81203	0,79871	0,7978	0,7984	0,7972	1,282	3,09
37	p37r5s6t7	0,79546	0,79752	0,79886	0,79735	0,7978	0,7984	0,7972	1,282	3,09
41	p41r5s6t7	0,79978	0,80106	0,79656	0,80109	0,7978	0,7984	0,7972	1,282	3,09
43	p43r5s6t7	0,79615	0,79557	0,79921	0,79939	0,7978	0,7984	0,7972	1,282	3,09
47	p47r5s6t7	0,79774	0,79954	0,79765	0,79849	0,7978	0,7984	0,7972	1,282	3,09
53	p53r5s6t7	0,79929	0,79572	0,79806	0,79924	0,7978	0,7984	0,7972	1,282	3,09
59	p59r5s6t7	0,79801	0,79455	0,79794	0,79726	0,7978	0,7984	0,7972	1,282	3,09
61	p61r5s6t7	0,79808	0,79549	0,79731	0,79457	0,7978	0,7984	0,7972	1,282	3,09
67	p67r5s6t7	0,7987	0,79497	0,80044	0,79281	0,7978	0,7984	0,7972	1,282	3,09
71	p71r5s6t7	0,79746	0,79992	0,79414	0,79533	0,7978	0,7984	0,7972	1,282	3,09
73	p73r5s6t7	0,79807	0,79817	0,79717	0,7991	0,7978	0,7984	0,7972	1,282	3,09
79	p79r5s6t7	0,7959	0,79572	0,79919	0,80134	0,7978	0,7984	0,7972	1,282	3,09
83	p83r5s6t7	0,79659	0,79913	0,79593	0,79715	0,7978	0,7984	0,7972	1,282	3,09
89	p89r5s6t7	0,79642	0,79322	0,7993	0,80014	0,7978	0,7984	0,7972	1,282	3,09
97	p97r5s6t7	0,79858	0,79799	0,79466	0,79731	0,7978	0,7984	0,7972	1,282	3,09
101	p101r5s6t7	0,79978	0,79942	0,794	0,79727	0,7978	0,7984	0,7972	1,282	3,09
103	p103r5s6t7	0,79316	0,80064	0,79994	0,79628	0,7978	0,7984	0,7972	1,282	3,09
107	p107r5s6t7	0,79821	0,79339	0,79442	0,79949	0,7978	0,7984	0,7972	1,282	3,09
109	p109r5s6t7	0,79593	0,79832	0,79609	0,7992	0,7978	0,7984	0,7972	1,282	3,09
113	p113r5s6t7	0,80162	0,79366	0,79331	0,79656	0,7978	0,7984	0,7972	1,282	3,09
127	p127r5s6t7	0,79772	0,79598	0,80052	0,79606	0,7978	0,7984	0,7972	1,282	3,09
131	p131r5s6t7	0,79602	0,79224	0,79768	0,79944	0,7978	0,7984	0,7972	1,282	3,09
137	p137r5s6t7	0,79943	0,79159	0,79792	0,79604	0,7978	0,7984	0,7972	1,282	3,09
139	p139r5s6t7	0,80061	0,79781	0,79705	0,7946	0,7978	0,7984	0,7972	1,282	3,09
149	p149r5s6t7	0,80033	0,79695	0,7967	0,7985	0,7978	0,7984	0,7972	1,282	3,09
151	p151r5s6t7	0,79741	0,79739	0,80017	0,79333	0,7978	0,7984	0,7972	1,282	3,09
157	p157r5s6t7	0,79454	0,79513	0,79477	0,79781	0,7978	0,7984	0,7972	1,282	3,09
163	p163r5s6t7	0,80068	0,79656	0,79851	0,80094	0,7978	0,7984	0,7972	1,282	3,09
167	p167r5s6t7	0,79724	0,79562	0,79955	0,79449	0,7978	0,7984	0,7972	1,282	3,09
173	p173r4s5t6	0,79607	0,79882	0,79864	0,7958	0,7978	0,7984	0,7972	1,282	3,09
179	p179r5s6t7	0,80135	0,80078	0,80111	0,79424	0,7978	0,7984	0,7972	1,282	3,09
181	p181r5s6t7	0,80015	0,7962	0,79563	0,79939	0,7978	0,7984	0,7972	1,282	3,09
191	p191r5s6t7	0,80361	0,79618	0,79365	0,79599	0,7978	0,7984	0,7972	1,282	3,09
193	p193r5s6t7	0,79441	0,80032	0,79901	0,80094	0,7978	0,7984	0,7972	1,282	3,09
197	p197r5s6t7	0,79649	0,79862	0,79768	0,79725	0,7978	0,7984	0,7972	1,282	3,09
199	p199r5s6t7	0,79503	0,7999	0,79184	0,79391	0,7978	0,7984	0,7972	1,282	3,09
211	p211r5s6t7	0,79947	0,79919	0,80095	0,79905	0,7978	0,7984	0,7972	1,282	3,09
223	p223r5s6t7	0,79694	0,79629	0,79851	0,80093	0,7978	0,7984	0,7972	1,282	3,09
227	p227r5s6t7	0,80487	0,79483	0,79069	0,80161	0,7978	0,7984	0,7972	1,282	3,09
229	p229r5s6t7	0,80258	0,79257	0,79829	0,79919	0,7978	0,7984	0,7972	1,282	3,09
233	p233r5s6t7	0,79292	0,79788	0,79772	0,79832	0,7978	0,7984	0,7972	1,282	3,09
239	p239r5s6t7	0,79562	0,79318	0,79665	0,79573	0,7978	0,7984	0,7972	1,282	3,09
241	p241r5s6t7	0,79602	0,79359	0,79701	0,80075	0,7978	0,7984	0,7972	1,282	3,09
251	p251r5s6t7	0,79359	0,79736	0,80279	0,79909	0,7978	0,7984	0,7972	1,282	3,09
257	p257r4s5t7	0,79707	0,80089	0,79652	0,79696	0,7978	0,7984	0,7972	1,282	3,09
263	p263r5s6t7	0,79493	0,7978	0,80097	0,79875	0,7978	0,7984	0,7972	1,282	3,09
269	p269r5s6t7	0,79607	0,80031	0,79969	0,79851	0,7978	0,7984	0,7972	1,282	3,09
271	p271r4s5t6	0,79585	0,79719	0,79754	0,79624	0,7978	0,7984	0,7972	1,282	3,09
277	p277r5s6t7	0,7972	0,7951	0,79741	0,79717	0,7978	0,7984	0,7972	1,282	3,09
281	p281r5s6t7	0,79976	0,79568	0,79866	0,79766	0,7978	0,7984	0,7972	1,282	3,09
283	p283r5s6t7	0,79841	0,79833	0,79651	0,79952	0,7978	0,7984	0,7972	1,282	3,09
293	p293r4s6t7	0,79533	0,7988	0,79267	0,80013	0,7978	0,7984	0,7972	1,282	3,09
307	p307r5s6t7	0,79573	0,79381	0,79765	0,79733	0,7978	0,7984	0,7972	1,282	3,09
311	p311r5s6t7	0,7985	0,79775	0,79799	0,79848	0,7978	0,7984	0,7972	1,282	3,09
313	p313r5s6t7	0,79327	0,79867	0,80066	0,7971	0,7978	0,7984	0,7972	1,282	3,09

Tabla del test de autocorrelación para combinaciones *min*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	0,79991	0,79869	13,19247	0,89943	0,7978	0,7984	0,7972	1,282	3,09
5	p5r3s5t6	0,79997	0,79221	4,93158	0,83203	0,7978	0,7984	0,7972	1,282	3,09
7	p7r3s4t5	0,7932	0,79694	2,60142	0,81575	0,7978	0,7984	0,7972	1,282	3,09
11	p11r2s4t5	0,80083	0,80016	1,14694	0,80749	0,7978	0,7984	0,7972	1,282	3,09
13	p13r2s4t5	0,79799	0,79917	1,02492	0,80471	0,7978	0,7984	0,7972	1,282	3,09
17	p17r2s4t5	0,79686	0,79873	0,87178	0,80326	0,7978	0,7984	0,7972	1,282	3,09
19	p19r2s4t5	0,79875	0,79384	0,8363	0,79555	0,7978	0,7984	0,7972	1,282	3,09
23	p23r2s3t5	0,79529	0,80116	0,82525	0,79573	0,7978	0,7984	0,7972	1,282	3,09
29	p29r2s3t5	0,80004	0,79747	0,80246	0,79597	0,7978	0,7984	0,7972	1,282	3,09
31	p31r2s3t4	0,80275	0,79753	0,80444	0,79491	0,7978	0,7984	0,7972	1,282	3,09
37	p37r2s3t4	0,79422	0,79769	0,80465	0,80053	0,7978	0,7984	0,7972	1,282	3,09
41	p41r2s3t4	0,79647	0,80139	0,8005	0,79392	0,7978	0,7984	0,7972	1,282	3,09
43	p43r2s3t4	0,79928	0,79704	0,7985	0,79562	0,7978	0,7984	0,7972	1,282	3,09
47	p47r2s3t4	0,80097	0,79684	0,80118	0,80027	0,7978	0,7984	0,7972	1,282	3,09
53	p53r2s3t4	0,7971	0,79976	0,79808	0,80145	0,7978	0,7984	0,7972	1,282	3,09
59	p59r2s3t4	0,79941	0,79696	0,79992	0,79968	0,7978	0,7984	0,7972	1,282	3,09
61	p61r2s3t4	0,79285	0,79782	0,79891	0,79655	0,7978	0,7984	0,7972	1,282	3,09
67	p67r2s3t4	0,79899	0,80182	0,79903	0,798	0,7978	0,7984	0,7972	1,282	3,09
71	p71r2s3t4	0,79952	0,79791	0,79446	0,79929	0,7978	0,7984	0,7972	1,282	3,09
73	p73r2s3t4	0,79719	0,79675	0,7974	0,79721	0,7978	0,7984	0,7972	1,282	3,09
79	p79r2s3t4	0,79636	0,80217	0,79699	0,79877	0,7978	0,7984	0,7972	1,282	3,09
83	p83r2s3t4	0,79683	0,79972	0,7974	0,80003	0,7978	0,7984	0,7972	1,282	3,09
89	p89r2s3t4	0,79842	0,79204	0,79989	0,80162	0,7978	0,7984	0,7972	1,282	3,09
97	p97r2s3t4	0,79619	0,79511	0,79737	0,80108	0,7978	0,7984	0,7972	1,282	3,09
101	p101r2s3t4	0,79663	0,79809	0,80055	0,80234	0,7978	0,7984	0,7972	1,282	3,09
103	p103r2s3t4	0,79689	0,80183	0,79885	0,79621	0,7978	0,7984	0,7972	1,282	3,09
107	p107r1s3t4	0,79906	0,7997	0,80029	0,79914	0,7978	0,7984	0,7972	1,282	3,09
109	p109r1s3t4	0,79746	0,79524	0,79632	0,79499	0,7978	0,7984	0,7972	1,282	3,09
113	p113r1s3t4	0,80187	0,79637	0,79665	0,795	0,7978	0,7984	0,7972	1,282	3,09
127	p127r1s3t4	0,79905	0,79759	0,79918	0,80014	0,7978	0,7984	0,7972	1,282	3,09
131	p131r1s3t4	0,79696	0,79769	0,79989	0,79798	0,7978	0,7984	0,7972	1,282	3,09
137	p137r1s3t4	0,80219	0,80058	0,79697	0,79807	0,7978	0,7984	0,7972	1,282	3,09
139	p139r1s3t4	0,79863	0,79412	0,79572	0,79958	0,7978	0,7984	0,7972	1,282	3,09
149	p149r1s3t4	0,79418	0,79868	0,79981	0,79663	0,7978	0,7984	0,7972	1,282	3,09
151	p151r1s3t4	0,80476	0,79384	0,79816	0,79652	0,7978	0,7984	0,7972	1,282	3,09
157	p157r1s3t4	0,79559	0,79882	0,79591	0,79606	0,7978	0,7984	0,7972	1,282	3,09
163	p163r1s3t4	0,79743	0,79882	0,79963	0,7952	0,7978	0,7984	0,7972	1,282	3,09
167	p167r1s3t4	0,79951	0,79987	0,79949	0,79679	0,7978	0,7984	0,7972	1,282	3,09
173	p173r1s3t4	0,80093	0,79753	0,79765	0,79445	0,7978	0,7984	0,7972	1,282	3,09
179	p179r1s3t4	0,8032	0,80177	0,79871	0,79407	0,7978	0,7984	0,7972	1,282	3,09
181	p181r1s3t4	0,79765	0,80195	0,79925	0,79966	0,7978	0,7984	0,7972	1,282	3,09
191	p191r1s3t4	0,7959	0,80005	0,7981	0,79706	0,7978	0,7984	0,7972	1,282	3,09
193	p193r1s3t4	0,80151	0,79847	0,7957	0,79778	0,7978	0,7984	0,7972	1,282	3,09
197	p197r1s3t4	0,804	0,80071	0,79458	0,80011	0,7978	0,7984	0,7972	1,282	3,09
199	p199r1s3t4	0,79887	0,79673	0,79617	0,79483	0,7978	0,7984	0,7972	1,282	3,09
211	p211r1s3t4	0,79436	0,79939	0,79775	0,79926	0,7978	0,7984	0,7972	1,282	3,09
223	p223r1s3t4	0,79731	0,79613	0,79809	0,79702	0,7978	0,7984	0,7972	1,282	3,09
227	p227r1s3t4	0,80032	0,79514	0,7949	0,79813	0,7978	0,7984	0,7972	1,282	3,09
229	p229r1s3t4	0,79631	0,79746	0,79703	0,79457	0,7978	0,7984	0,7972	1,282	3,09
233	p233r1s3t4	0,80279	0,79925	0,7995	0,7938	0,7978	0,7984	0,7972	1,282	3,09
239	p239r1s3t4	0,79793	0,79585	0,79998	0,79777	0,7978	0,7984	0,7972	1,282	3,09
241	p241r1s3t4	0,80226	0,79364	0,79529	0,79876	0,7978	0,7984	0,7972	1,282	3,09
251	p251r1s3t4	0,79813	0,79575	0,79955	0,79621	0,7978	0,7984	0,7972	1,282	3,09
257	p257r1s3t4	0,79971	0,80251	0,79908	0,79801	0,7978	0,7984	0,7972	1,282	3,09
263	p263r1s3t4	0,7999	0,80151	0,79667	0,79785	0,7978	0,7984	0,7972	1,282	3,09
269	p269r1s3t4	0,79615	0,80264	0,7956	0,79063	0,7978	0,7984	0,7972	1,282	3,09
271	p271r1s3t4	0,79616	0,79866	0,7924	0,79541	0,7978	0,7984	0,7972	1,282	3,09
277	p277r1s3t4	0,79969	0,79838	0,79912	0,80235	0,7978	0,7984	0,7972	1,282	3,09
281	p281r1s3t4	0,79511	0,79455	0,80165	0,79832	0,7978	0,7984	0,7972	1,282	3,09
283	p283r1s3t4	0,79841	0,79913	0,80005	0,79667	0,7978	0,7984	0,7972	1,282	3,09
293	p293r1s3t4	0,8	0,79869	0,80201	0,7976	0,7978	0,7984	0,7972	1,282	3,09
307	p307r1s3t4	0,80294	0,79614	0,80297	0,79924	0,7978	0,7984	0,7972	1,282	3,09
311	p311r1s3t4	0,79896	0,79614	0,79378	0,79887	0,7978	0,7984	0,7972	1,282	3,09
313	p313r1s3t4	0,79655	0,80318	0,80061	0,7971	0,7978	0,7984	0,7972	1,282	3,09

Tabla del test de autocorrelación para combinaciones *bottom*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	α=0,1	α=0,001
3	p3r4s5t7	0,79991	0,79869	13,19247	0,89943	0,7978	0,7984	0,7972	1,282	3,09
5	p5r2s5t7	0,80032	0,80102	4,8579	0,82999	0,7978	0,7984	0,7972	1,282	3,09
7	p7r1s5t7	0,79534	0,79676	2,58929	0,80963	0,7978	0,7984	0,7972	1,282	3,09
11	p11r1s4t7	0,79453	0,79347	1,16449	0,80425	0,7978	0,7984	0,7972	1,282	3,09
13	p13r1s5t6	0,7981	0,79597	0,9486	0,79711	0,7978	0,7984	0,7972	1,282	3,09
17	p17r1s4t6	0,80064	0,79823	0,87507	0,80097	0,7978	0,7984	0,7972	1,282	3,09
19	p19r1s3t7	0,79672	0,79618	0,84401	0,79966	0,7978	0,7984	0,7972	1,282	3,09
23	p23r1s3t7	0,80005	0,80219	0,82431	0,79469	0,7978	0,7984	0,7972	1,282	3,09
29	p29r1s3t7	0,79522	0,79743	0,80063	0,803	0,7978	0,7984	0,7972	1,282	3,09
31	p31r1s3t7	0,79816	0,79276	0,79886	0,79834	0,7978	0,7984	0,7972	1,282	3,09
37	p37r1s3t7	0,80208	0,79516	0,80398	0,79948	0,7978	0,7984	0,7972	1,282	3,09
41	p41r1s3t7	0,79701	0,79692	0,80315	0,796	0,7978	0,7984	0,7972	1,282	3,09
43	p43r1s3t7	0,798	0,80156	0,80191	0,79766	0,7978	0,7984	0,7972	1,282	3,09
47	p47r1s3t7	0,80057	0,79883	0,80122	0,79593	0,7978	0,7984	0,7972	1,282	3,09
53	p53r1s2t7	0,80209	0,80149	0,79397	0,79911	0,7978	0,7984	0,7972	1,282	3,09
59	p59r1s3t5	0,79597	0,79756	0,80029	0,80121	0,7978	0,7984	0,7972	1,282	3,09
61	p61r1s2t7	0,79705	0,80025	0,80068	0,79853	0,7978	0,7984	0,7972	1,282	3,09
67	p67r1s3t5	0,79644	0,79909	0,79694	0,79975	0,7978	0,7984	0,7972	1,282	3,09
71	p71r1s3t5	0,79626	0,79944	0,80182	0,79532	0,7978	0,7984	0,7972	1,282	3,09
73	p73r1s2t7	0,79647	0,80172	0,79873	0,79938	0,7978	0,7984	0,7972	1,282	3,09
79	p79r1s3t5	0,79548	0,80062	0,79918	0,80149	0,7978	0,7984	0,7972	1,282	3,09
83	p83r1s3t5	0,79679	0,79862	0,7973	0,79601	0,7978	0,7984	0,7972	1,282	3,09
89	p89r1s3t5	0,80011	0,79519	0,79782	0,79998	0,7978	0,7984	0,7972	1,282	3,09
97	p97r1s2t7	0,80103	0,7942	0,79607	0,80008	0,7978	0,7984	0,7972	1,282	3,09
101	p101r1s3t5	0,79718	0,79545	0,79466	0,79914	0,7978	0,7984	0,7972	1,282	3,09
103	p103r1s2t7	0,79864	0,80129	0,79662	0,7959	0,7978	0,7984	0,7972	1,282	3,09
107	p107r1s3t4	0,79906	0,7997	0,80029	0,79914	0,7978	0,7984	0,7972	1,282	3,09
109	p109r1s3t4	0,79746	0,79524	0,79632	0,79499	0,7978	0,7984	0,7972	1,282	3,09
113	p113r1s3t4	0,80187	0,79637	0,79665	0,795	0,7978	0,7984	0,7972	1,282	3,09
127	p127r1s3t4	0,79905	0,79759	0,79918	0,80014	0,7978	0,7984	0,7972	1,282	3,09
131	p131r1s3t4	0,79696	0,79769	0,79989	0,79798	0,7978	0,7984	0,7972	1,282	3,09
137	p137r1s2t7	0,79193	0,7984	0,79544	0,79941	0,7978	0,7984	0,7972	1,282	3,09
139	p139r1s3t4	0,79863	0,79412	0,79572	0,79958	0,7978	0,7984	0,7972	1,282	3,09
149	p149r1s3t4	0,79418	0,79868	0,79981	0,79663	0,7978	0,7984	0,7972	1,282	3,09
151	p151r1s2t7	0,80044	0,79858	0,79803	0,79526	0,7978	0,7984	0,7972	1,282	3,09
157	p157r1s2t7	0,79542	0,79873	0,79576	0,79839	0,7978	0,7984	0,7972	1,282	3,09
163	p163r1s2t7	0,79791	0,79969	0,79943	0,79836	0,7978	0,7984	0,7972	1,282	3,09
167	p167r1s3t4	0,79951	0,79987	0,79949	0,79679	0,7978	0,7984	0,7972	1,282	3,09
173	p173r1s3t4	0,80093	0,79753	0,79765	0,79445	0,7978	0,7984	0,7972	1,282	3,09
179	p179r1s3t4	0,8032	0,80177	0,79871	0,79407	0,7978	0,7984	0,7972	1,282	3,09
181	p181r1s3t4	0,79765	0,80195	0,79925	0,79966	0,7978	0,7984	0,7972	1,282	3,09
191	p191r1s3t4	0,7959	0,80005	0,7981	0,79706	0,7978	0,7984	0,7972	1,282	3,09
193	p193r1s2t5	0,79689	0,79874	0,79433	0,79736	0,7978	0,7984	0,7972	1,282	3,09
197	p197r1s3t4	0,804	0,80071	0,79458	0,80011	0,7978	0,7984	0,7972	1,282	3,09
199	p199r1s3t4	0,79887	0,79673	0,79617	0,79483	0,7978	0,7984	0,7972	1,282	3,09
211	p211r1s2t5	0,80044	0,80399	0,7956	0,79349	0,7978	0,7984	0,7972	1,282	3,09
223	p223r1s3t4	0,79731	0,79613	0,79809	0,79702	0,7978	0,7984	0,7972	1,282	3,09
227	p227r1s3t4	0,80032	0,79514	0,7949	0,79813	0,7978	0,7984	0,7972	1,282	3,09
229	p229r1s2t5	0,79794	0,79929	0,79879	0,79794	0,7978	0,7984	0,7972	1,282	3,09
233	p233r1s3t4	0,80279	0,79925	0,7995	0,7938	0,7978	0,7984	0,7972	1,282	3,09
239	p239r1s3t4	0,79793	0,79585	0,79998	0,79777	0,7978	0,7984	0,7972	1,282	3,09
241	p241r1s2t5	0,7954	0,79216	0,80129	0,79498	0,7978	0,7984	0,7972	1,282	3,09
251	p251r1s3t4	0,79813	0,79575	0,79955	0,79621	0,7978	0,7984	0,7972	1,282	3,09
257	p257r1s2t5	0,79871	0,80034	0,79772	0,7998	0,7978	0,7984	0,7972	1,282	3,09
263	p263r1s2t5	0,80031	0,79414	0,79654	0,79876	0,7978	0,7984	0,7972	1,282	3,09
269	p269r1s3t4	0,79615	0,80264	0,7956	0,79063	0,7978	0,7984	0,7972	1,282	3,09
271	p271r1s2t5	0,79762	0,79697	0,79818	0,79787	0,7978	0,7984	0,7972	1,282	3,09
277	p277r1s2t5	0,79867	0,80166	0,79865	0,79718	0,7978	0,7984	0,7972	1,282	3,09
281	p281r1s2t5	0,79823	0,79825	0,79922	0,79444	0,7978	0,7984	0,7972	1,282	3,09
283	p283r1s2t5	0,79782	0,79826	0,79493	0,79808	0,7978	0,7984	0,7972	1,282	3,09
293	p293r1s2t5	0,80028	0,79643	0,79809	0,79677	0,7978	0,7984	0,7972	1,282	3,09
307	p307r1s3t4	0,80294	0,79614	0,80297	0,79924	0,7978	0,7984	0,7972	1,282	3,09
311	p311r1s3t4	0,79896	0,79614	0,79378	0,79887	0,7978	0,7984	0,7972	1,282	3,09
313	p313r1s2t5	0,79989	0,79871	0,79555	0,79483	0,7978	0,7984	0,7972	1,282	3,09

Tabla de complejidad lineal para combinaciones *top*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	Min.	Max.
3	p3r5s6t7	10000,3	10000,5	10000,2	10000,1	10000	10000	10000	9999	10001
5	p5r5s6t7	9999,8	10000,2	10000,3	9999,8	10000	10000	10000	9999	10001
7	p7r5s6t7	10000,1	10000,3	9999,7	10000,1	10000	10000	10000	9999	10001
11	p11r5s6t7	10000,3	9999,9	10000,1	10000,5	10000	10000	10000	9999	10001
13	p13r5s6t7	10000,6	10000	10000,3	10000	10000	10000	10000	9999	10001
17	p17r5s6t7	10000	10000,5	10000,3	9999,9	10000	10000	10000	9999	10001
19	p19r5s6t7	10000,1	10000,3	10000,3	9999,9	10000	10000	10000	9999	10001
23	p23r5s6t7	10000,1	10000,2	10000,4	9999,7	10000	10000	10000	9999	10001
29	p29r5s6t7	9999,8	10000,5	10000,8	10000,6	10000	10000	10000	9999	10001
31	p31r5s6t7	10000	10000,2	9999,9	10000,5	10000	10000	10000	9999	10001
37	p37r5s6t7	10000	10000,5	9999,7	10000,3	10000	10000	10000	9999	10001
41	p41r5s6t7	9999,8	9999,5	10000,3	10000,2	10000	10000	10000	9999	10001
43	p43r5s6t7	10000,3	10000,5	10000,9	10000,3	10000	10000	10000	9999	10001
47	p47r5s6t7	10000,6	10000	10000,3	9999,8	10000	10000	10000	9999	10001
53	p53r5s6t7	10000,6	10000,5	10000,2	10000,9	10000	10000	10000	9999	10001
59	p59r5s6t7	10000,3	10000,1	9999,9	9999,6	10000	10000	10000	9999	10001
61	p61r5s6t7	10000,2	10000,4	10000,5	10000,1	10000	10000	10000	9999	10001
67	p67r5s6t7	10000,7	10000,5	10000	10000,5	10000	10000	10000	9999	10001
71	p71r5s6t7	10000,5	10000,5	9999,8	9999,9	10000	10000	10000	9999	10001
73	p73r5s6t7	9999,7	10001,2	10000,1	10000,3	10000	10000	10000	9999	10001
79	p79r5s6t7	10000	10000	10000,8	10000,4	10000	10000	10000	9999	10001
83	p83r5s6t7	10000,8	10000	10000,6	10000,2	10000	10000	10000	9999	10001
89	p89r5s6t7	10000,7	10000,2	10000,4	10000,5	10000	10000	10000	9999	10001
97	p97r5s6t7	9999,5	10000,2	10000,4	10000,6	10000	10000	10000	9999	10001
101	p101r5s6t7	10000,1	10000,5	10000,3	10000,3	10000	10000	10000	9999	10001
103	p103r5s6t7	9999,9	10000	10000,1	9999,9	10000	10000	10000	9999	10001
107	p107r5s6t7	10000,2	10000,8	10001	9999,8	10000	10000	10000	9999	10001
109	p109r5s6t7	10000,2	10000,3	9999,7	10000,7	10000	10000	10000	9999	10001
113	p113r5s6t7	10000,1	10000,3	10000,2	10000,1	10000	10000	10000	9999	10001
127	p127r5s6t7	9999,9	9999,6	10000,5	10000,2	10000	10000	10000	9999	10001
131	p131r5s6t7	9999,1	10000,3	10000,5	10000,4	10000	10000	10000	9999	10001
137	p137r5s6t7	10000,2	10000	10000,5	9999,8	10000	10000	10000	9999	10001
139	p139r5s6t7	9999,9	10000,3	10000,3	10000,4	10000	10000	10000	9999	10001
149	p149r5s6t7	10000,5	9999,7	10000,4	9999,9	10000	10000	10000	9999	10001
151	p151r5s6t7	10000,1	9999,9	10000,5	10000,2	10000	10000	10000	9999	10001
157	p157r5s6t7	10000,6	10000,5	10000,2	10000,3	10000	10000	10000	9999	10001
163	p163r5s6t7	10000	10000	10000,3	9999,7	10000	10000	10000	9999	10001
167	p167r5s6t7	10000,4	10000,3	10000,2	10000,1	10000	10000	10000	9999	10001
173	p173r4s5t6	9999,6	10000,5	9999,9	10000,5	10000	10000	10000	9999	10001
179	p179r5s6t7	10000,1	9999,9	10000	10000	10000	10000	10000	9999	10001
181	p181r5s6t7	10000,2	10000,3	10000,3	10000	10000	10000	10000	9999	10001
191	p191r5s6t7	10000,3	10000,6	10000,7	10000,2	10000	10000	10000	9999	10001
193	p193r5s6t7	10000,4	10000,3	9999,7	10000,2	10000	10000	10000	9999	10001
197	p197r5s6t7	9999,9	10000,2	9999,9	10000,3	10000	10000	10000	9999	10001
199	p199r5s6t7	10000,4	10000	10000,1	10000,9	10000	10000	10000	9999	10001
211	p211r5s6t7	9999,8	10000,4	10000,9	10000,2	10000	10000	10000	9999	10001
223	p223r5s6t7	10000,2	9999,7	9999,8	10000,5	10000	10000	10000	9999	10001
227	p227r5s6t7	9999,9	10000,1	10000,8	10000,2	10000	10000	10000	9999	10001
229	p229r5s6t7	10000	10000,7	9999,8	10000,8	10000	10000	10000	9999	10001
233	p233r5s6t7	9999,9	10000,5	9999,8	10000,6	10000	10000	10000	9999	10001
239	p239r5s6t7	10000	10000,1	10000,6	10000,5	10000	10000	10000	9999	10001
241	p241r5s6t7	10000,2	9999,7	10000,1	10000,3	10000	10000	10000	9999	10001
251	p251r5s6t7	10000	10000,2	10000,2	10000,6	10000	10000	10000	9999	10001
257	p257r4s5t7	10000,6	10000,3	10000,1	10000,1	10000	10000	10000	9999	10001
263	p263r5s6t7	10000,9	9999,9	10000,3	10000,3	10000	10000	10000	9999	10001
269	p269r5s6t7	10000,1	10000,2	10000,2	9999,8	10000	10000	10000	9999	10001
271	p271r4s5t6	10000	10000,6	10000,3	9999,8	10000	10000	10000	9999	10001
277	p277r5s6t7	9999,6	10000,7	10000,2	10000,2	10000	10000	10000	9999	10001
281	p281r5s6t7	9999,9	10000,2	10000,1	10000,5	10000	10000	10000	9999	10001
283	p283r5s6t7	10000,6	10000,2	10000,4	10000,5	10000	10000	10000	9999	10001
293	p293r4s6t7	10000,6	10000,3	10000,1	10000,5	10000	10000	10000	9999	10001
307	p307r5s6t7	10000,4	10000	10000,2	10000,3	10000	10000	10000	9999	10001
311	p311r5s6t7	10000,2	10000,3	10000,1	10000,3	10000	10000	10000	9999	10001
313	p313r5s6t7	10000	9999,9	10000,2	10000,1	10000	10000	10000	9999	10001

Tabla de complejidad lineal para combinaciones *min*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	Min.	Max.
3	p3r4s5t7	10000,3	9999,9	10000,7	10000,6	10000	10000	10000	9999	10001
5	p5r3s5t6	10000,4	10000,1	10000,6	10000,2	10000	10000	10000	9999	10001
7	p7r3s4t5	10000,3	10000,1	10000,4	10000,1	10000	10000	10000	9999	10001
11	p11r2s4t5	10000	10000,5	10000	10000,4	10000	10000	10000	9999	10001
13	p13r2s4t5	10000,6	10000,6	9999,6	10000,6	10000	10000	10000	9999	10001
17	p17r2s4t5	10000	10000	10000,2	10000,4	10000	10000	10000	9999	10001
19	p19r2s4t5	9999,9	10000,4	10000,1	10000,1	10000	10000	10000	9999	10001
23	p23r2s3t5	10000,2	10000,3	9999,6	10000,6	10000	10000	10000	9999	10001
29	p29r2s3t5	10000,2	10000,6	10000,2	10000,1	10000	10000	10000	9999	10001
31	p31r2s3t4	10000,1	10000	10000,6	10000,4	10000	10000	10000	9999	10001
37	p37r2s3t4	9999,9	10000,4	10000,2	10000,6	10000	10000	10000	9999	10001
41	p41r2s3t4	10000,6	10000,5	9999,7	10000,6	10000	10000	10000	9999	10001
43	p43r2s3t4	10000,4	10000,1	10000,1	10000	10000	10000	10000	9999	10001
47	p47r2s3t4	9999,7	10000,4	9999,8	10000,5	10000	10000	10000	9999	10001
53	p53r2s3t4	9999,9	10000,3	10000,5	10000,9	10000	10000	10000	9999	10001
59	p59r2s3t4	10000,1	10000,8	9999,8	9999,8	10000	10000	10000	9999	10001
61	p61r2s3t4	10000,2	10000,3	10000,1	10000,3	10000	10000	10000	9999	10001
67	p67r2s3t4	9999,8	10000,6	10000,8	10000,3	10000	10000	10000	9999	10001
71	p71r2s3t4	9999,9	10000,4	10000,5	10000,9	10000	10000	10000	9999	10001
73	p73r2s3t4	10000,4	10000,3	9999,9	10000,1	10000	10000	10000	9999	10001
79	p79r2s3t4	9999,6	10000,6	9999,8	10000,4	10000	10000	10000	9999	10001
83	p83r2s3t4	10000,5	10000,4	10000,1	10000,1	10000	10000	10000	9999	10001
89	p89r2s3t4	10000,2	10000,6	9999,7	10000,1	10000	10000	10000	9999	10001
97	p97r2s3t4	10000,2	10000,5	10000,2	10000,2	10000	10000	10000	9999	10001
101	p101r2s3t4	10000,1	10000,1	10000,8	10000,6	10000	10000	10000	9999	10001
103	p103r2s3t4	10000,4	10000,4	9999,7	10000,2	10000	10000	10000	9999	10001
107	p107r1s3t4	10000,7	10000	10000,5	10000	10000	10000	10000	9999	10001
109	p109r1s3t4	10000,4	9999,9	9999,9	10000,4	10000	10000	10000	9999	10001
113	p113r1s3t4	10000	10000	10000,4	10000,5	10000	10000	10000	9999	10001
127	p127r1s3t4	10000,2	10000	10000,2	10000,4	10000	10000	10000	9999	10001
131	p131r1s3t4	10000,6	10000,3	10000	10000,5	10000	10000	10000	9999	10001
137	p137r1s3t4	10000,7	9999,9	10000,4	10000,6	10000	10000	10000	9999	10001
139	p139r1s3t4	10000,2	10000,2	10000,4	10000,1	10000	10000	10000	9999	10001
149	p149r1s3t4	10000,4	10000,1	9999,6	10000,7	10000	10000	10000	9999	10001
151	p151r1s3t4	10000,4	10000,1	10000,2	10000,5	10000	10000	10000	9999	10001
157	p157r1s3t4	9999,9	10001	10000,5	10000,7	10000	10000	10000	9999	10001
163	p163r1s3t4	10000,7	10000	9999,7	10000,2	10000	10000	10000	9999	10001
167	p167r1s3t4	10000,4	10000,3	10000,1	10000,9	10000	10000	10000	9999	10001
173	p173r1s3t4	10000,7	10000,1	10000,1	10000,2	10000	10000	10000	9999	10001
179	p179r1s3t4	10000,8	10000,3	10000,5	10000,6	10000	10000	10000	9999	10001
181	p181r1s3t4	10000,5	10000	10000,7	10000,9	10000	10000	10000	9999	10001
191	p191r1s3t4	10000,2	10000	9999,9	10000,2	10000	10000	10000	9999	10001
193	p193r1s3t4	10000,1	10000,5	10000,2	10000,2	10000	10000	10000	9999	10001
197	p197r1s3t4	10000,2	10000	10000	9999,8	10000	10000	10000	9999	10001
199	p199r1s3t4	10000,3	9999,8	10000,3	10000,1	10000	10000	10000	9999	10001
211	p211r1s3t4	10000,3	10000,3	10000,7	9999,6	10000	10000	10000	9999	10001
223	p223r1s3t4	10000,9	10000,3	10000,5	9999,6	10000	10000	10000	9999	10001
227	p227r1s3t4	10000,1	10000,8	10000,8	9999,9	10000	10000	10000	9999	10001
229	p229r1s3t4	10000	10000,6	10000,4	10000,7	10000	10000	10000	9999	10001
233	p233r1s3t4	10000,7	9999,8	10000,4	9999,9	10000	10000	10000	9999	10001
239	p239r1s3t4	10000,7	10000,1	10000	10000,3	10000	10000	10000	9999	10001
241	p241r1s3t4	10000,5	9999,5	10000,6	10000,3	10000	10000	10000	9999	10001
251	p251r1s3t4	10000,4	10000,5	10000,2	10000,1	10000	10000	10000	9999	10001
257	p257r1s3t4	10000,4	10000,1	10000,5	9999,7	10000	10000	10000	9999	10001
263	p263r1s3t4	10000,3	10000,4	9999,9	10000,4	10000	10000	10000	9999	10001
269	p269r1s3t4	9999,7	10000,5	10000,1	10000,5	10000	10000	10000	9999	10001
271	p271r1s3t4	10000,6	9999,7	9999,8	10000,3	10000	10000	10000	9999	10001
277	p277r1s3t4	10000,5	10000,3	10000,3	10000,5	10000	10000	10000	9999	10001
281	p281r1s3t4	10000,2	10000,9	10000,4	10000,7	10000	10000	10000	9999	10001
283	p283r1s3t4	10000,2	10000,1	10000,1	10000,7	10000	10000	10000	9999	10001
293	p293r1s3t4	9999,8	10000,2	10000	10000	10000	10000	10000	9999	10001
307	p307r1s3t4	9999,9	9999,7	9999,9	10000,4	10000	10000	10000	9999	10001
311	p311r1s3t4	10000,4	9999,9	9999,6	10000,2	10000	10000	10000	9999	10001
313	p313r1s3t4	10000,3	10000,5	10000,2	9999,7	10000	10000	10000	9999	10001

Tabla de complejidad lineal para combinaciones *bottom*

Primo	Combinación	*slow-n*	*slow-f*	*fast-n*	*fast-f*	BBS	AES	RC4	Min.	Max.
3	p3r4s5t7	10000,3	9999,9	10000,7	10000,6	10000	10000	10000	9999	10001
5	p5r2s5t7	10000,4	10000,4	9999,5	10000,4	10000	10000	10000	9999	10001
7	p7r1s5t7	9999,7	10000,3	10000,3	10000,3	10000	10000	10000	9999	10001
11	p11r1s4t7	10000,8	10000	10000,3	10000,6	10000	10000	10000	9999	10001
13	p13r1s5t6	10000,4	10000,4	10000,5	9999,8	10000	10000	10000	9999	10001
17	p17r1s4t6	10000	10000,3	10000,4	10000,2	10000	10000	10000	9999	10001
19	p19r1s3t7	9999,7	10000,2	10000,7	10000,6	10000	10000	10000	9999	10001
23	p23r1s3t7	9999,8	10000	10000,2	9999,7	10000	10000	10000	9999	10001
29	p29r1s3t7	10000,2	10000,2	10000,3	10000,3	10000	10000	10000	9999	10001
31	p31r1s3t7	10000,5	10000,5	10000,1	10000,1	10000	10000	10000	9999	10001
37	p37r1s3t7	10000,6	10000	10000,3	10000	10000	10000	10000	9999	10001
41	p41r1s3t7	10000,4	10000,4	9999,6	9999,5	10000	10000	10000	9999	10001
43	p43r1s3t7	10000,7	10000,1	9999,8	9999,6	10000	10000	10000	9999	10001
47	p47r1s3t7	9999,7	10000,4	10000,3	10000,1	10000	10000	10000	9999	10001
53	p53r1s2t7	10000,7	10000	10000,3	9999,3	10000	10000	10000	9999	10001
59	p59r1s3t5	10000,1	10000	9999,9	10000	10000	10000	10000	9999	10001
61	p61r1s2t7	10000,5	9999,6	10000,3	10000,6	10000	10000	10000	9999	10001
67	p67r1s3t5	10000,5	10000,5	10000,6	10000,6	10000	10000	10000	9999	10001
71	p71r1s3t5	10000,1	9999,8	10000,2	9999,9	10000	10000	10000	9999	10001
73	p73r1s2t7	9999,3	10000,6	9999,9	10000,4	10000	10000	10000	9999	10001
79	p79r1s3t5	10000	9999,9	10000,4	9999,8	10000	10000	10000	9999	10001
83	p83r1s3t5	10000,1	10000,7	10000	9999,8	10000	10000	10000	9999	10001
89	p89r1s3t5	9999,9	10000,3	10001,3	10000,2	10000	10000	10000	9999	10001
97	p97r1s2t7	10000,2	10000,5	10000	10000,8	10000	10000	10000	9999	10001
101	p101r1s3t5	10000,5	9999,7	10000	10000,6	10000	10000	10000	9999	10001
103	p103r1s2t7	10000,3	10000,1	10000,6	10000,1	10000	10000	10000	9999	10001
107	p107r1s3t4	10000,7	10000	10000,5	10000	10000	10000	10000	9999	10001
109	p109r1s3t4	10000,4	9999,9	9999,9	10000,4	10000	10000	10000	9999	10001
113	p113r1s3t4	10000	10000	10000,4	10000,5	10000	10000	10000	9999	10001
127	p127r1s3t4	10000,2	10000	10000,2	10000,4	10000	10000	10000	9999	10001
131	p131r1s3t4	10000,6	10000,3	10000	10000,5	10000	10000	10000	9999	10001
137	p137r1s2t7	10000,7	10000,4	10000,3	10000,7	10000	10000	10000	9999	10001
139	p139r1s3t4	10000,2	10000,2	10000,4	10000,1	10000	10000	10000	9999	10001
149	p149r1s3t4	10000,4	10000,1	9999,6	10000,7	10000	10000	10000	9999	10001
151	p151r1s2t7	10000	10000,9	9999,6	10000,1	10000	10000	10000	9999	10001
157	p157r1s2t7	10000,1	10001,1	10000,9	10000,3	10000	10000	10000	9999	10001
163	p163r1s2t7	9999,9	10000,3	10000,5	10000,3	10000	10000	10000	9999	10001
167	p167r1s3t4	10000,4	10000,3	10000,1	10000,9	10000	10000	10000	9999	10001
173	p173r1s3t4	10000,7	10000,1	10000,1	10000,2	10000	10000	10000	9999	10001
179	p179r1s3t4	10000,8	10000,3	10000,5	10000,6	10000	10000	10000	9999	10001
181	p181r1s3t4	10000,5	10000	10000,7	10000,9	10000	10000	10000	9999	10001
191	p191r1s3t4	10000,2	10000	9999,9	10000,2	10000	10000	10000	9999	10001
193	p193r1s2t5	9999,6	10000	10000,1	10000,5	10000	10000	10000	9999	10001
197	p197r1s3t4	10000,2	10000	10000	9999,8	10000	10000	10000	9999	10001
199	p199r1s3t4	10000,3	9999,8	10000,3	10000,1	10000	10000	10000	9999	10001
211	p211r1s2t5	9999,8	10000,4	10000,6	10000	10000	10000	10000	9999	10001
223	p223r1s3t4	10000,9	10000,3	10000,5	9999,6	10000	10000	10000	9999	10001
227	p227r1s3t4	10000,1	10000,8	10000,8	9999,9	10000	10000	10000	9999	10001
229	p229r1s2t5	10000,2	10000,3	10000,4	10000,1	10000	10000	10000	9999	10001
233	p233r1s3t4	10000,7	9999,8	10000,4	9999,9	10000	10000	10000	9999	10001
239	p239r1s3t4	10000,7	10000,1	10000	10000,3	10000	10000	10000	9999	10001
241	p241r1s2t5	10000	10000,7	10000	10000,4	10000	10000	10000	9999	10001
251	p251r1s3t4	10000,4	10000,5	10000,2	10000,1	10000	10000	10000	9999	10001
257	p257r1s2t5	10000,3	10000,2	10000,1	10000,7	10000	10000	10000	9999	10001
263	p263r1s2t5	10000,4	10000,1	10000,2	10000,4	10000	10000	10000	9999	10001
269	p269r1s3t4	9999,7	10000,5	10000,1	10000,5	10000	10000	10000	9999	10001
271	p271r1s2t5	10000,7	10000,5	9999,9	10000,6	10000	10000	10000	9999	10001
277	p277r1s2t5	10000,9	10000,3	10000,6	10000,3	10000	10000	10000	9999	10001
281	p281r1s2t5	10000,5	10000,3	10000,4	10000,5	10000	10000	10000	9999	10001
283	p283r1s2t5	10000,1	10000,2	9999,9	10000,2	10000	10000	10000	9999	10001
293	p293r1s2t5	10000,2	10000,2	10000,1	10000,3	10000	10000	10000	9999	10001
307	p307r1s3t4	9999,9	9999,7	9999,9	10000,4	10000	10000	10000	9999	10001
311	p311r1s3t4	10000,4	9999,9	9999,6	10000,2	10000	10000	10000	9999	10001
313	p313r1s2t5	10000,8	10000,5	10000,4	10000,3	10000	10000	10000	9999	10001

Ciclos de CPU

				top							*min*							*bottom*			
				slow		*fast*					*slow*		*fast*					*slow*		*fast*	
p	**r**	**s**	**t**	**no fil.**	**filtro**	**no fil.**	**filtro**	**r**	**s**	**t**	**no fil.**	**filtro**	**no fil.**	**filtro**	**r**	**s**	**t**	**no fil.**	**filtro**	**no fil.**	**filtro**
3	5	6	7	2014372	2013683	18822	18969	4	5	7	1455878	1460436	17711	17834	4	5	7	1502489	1475298	17711	17864
5	5	6	7	2026439	2014125	18825	18935	3	5	6	1111569	1115811	17681	17796	2	5	7	1047237	1051170	17866	17976
7	5	6	7	2024583	2013465	18832	18935	3	4	5	824553	832619	17619	17742	1	5	7	877741	880721	18815	18937
11	5	6	7	2030545	2011242	18830	18941	2	4	5	675824	676303	17959	18062	1	4	7	711825	703206	18262	18541
13	5	6	7	2003550	1995505	18839	18947	2	4	5	672959	672903	18043	18148	1	5	6	781084	807329	19156	19223
17	5	6	7	2001168	2001623	18824	18938	2	4	5	674156	670562	17990	18090	1	4	6	626469	623302	18636	18746
19	5	6	7	1997143	1996845	18821	18936	2	4	5	673524	672841	18022	18130	1	3	7	551065	549925	18185	18298
23	5	6	7	1999303	1997227	18837	18937	2	3	5	555118	546986	18213	18332	1	3	7	555079	553409	18082	18307
29	5	6	7	1999394	2005516	18826	18938	2	3	5	545705	553576	18215	18330	1	3	7	552719	554235	18183	18294
31	5	6	7	1995858	2004754	18826	18942	2	3	4	485923	484864	19229	19338	1	3	7	551261	556932	18185	18293
37	5	6	7	2008896	1998368	18823	18938	2	3	4	488689	488493	19299	19129	1	3	7	553642	553166	18181	18273
41	5	6	7	2000227	2001281	18821	18937	2	3	4	487413	487026	19219	19346	1	3	7	555625	553795	18184	18301
43	5	6	7	2006909	2008633	18838	18931	2	3	4	486337	481520	19228	19344	1	3	7	555380	558248	18185	18301
47	5	6	7	2018654	2004849	18839	18940	2	3	4	487351	486393	19226	19338	1	3	7	552905	556294	18184	18291
53	5	6	7	2013712	2001323	18839	18940	2	3	4	485409	482883	19114	19338	1	2	7	430149	429190	18863	18970
59	5	6	7	2008446	1997092	18908	18940	2	3	4	485369	486613	19029	19337	1	3	5	439690	441647	19564	19713
61	5	6	7	2012757	2009636	18842	18941	2	3	4	487225	485294	19159	19338	1	2	7	427482	429403	18880	19036
67	5	6	7	2006973	2011646	18824	18941	2	3	4	496412	493243	19227	19184	1	3	5	440754	443315	19589	19698
71	5	6	7	2007467	2000614	18836	18941	2	3	4	483510	486637	19226	19231	1	3	5	438690	438379	19589	19696
73	5	6	7	2000728	2003270	18842	18940	2	3	4	483915	485515	19227	19337	1	2	7	429288	429260	18886	19027
79	5	6	7	2006861	2001646	18825	18938	2	3	4	487572	487749	19264	19345	1	3	5	443521	438814	19589	19679
83	5	6	7	2013665	2014067	18839	19115	2	3	4	486896	484296	19217	19342	1	3	5	443241	442303	19590	19695
89	5	6	7	2002928	2013002	18838	18938	2	3	4	487774	488870	19226	19337	1	3	5	442101	440650	19591	19698
97	5	6	7	1999985	2005969	18843	18942	2	3	4	487186	488301	19083	19343	1	2	7	430254	427878	18870	18959
101	5	6	7	2008125	2008138	18828	18942	2	3	4	487207	481279	19324	19310	1	3	5	443707	444528	19573	19699
103	5	6	7	2015479	2013913	18835	18940	2	3	4	485305	487270	19224	19273	1	2	7	428867	427197	18952	18969
107	5	6	7	2013755	2013348	18823	18938	1	3	4	393879	395151	21278	21247	1	3	4	393513	390931	21039	21104
109	5	6	7	2007001	2014702	18825	18942	1	3	4	388209	392320	21083	21094	1	3	4	390108	398184	21101	21158
113	5	6	7	2003336	2011855	18839	18942	1	3	4	393169	395269	21090	21159	1	3	4	390723	392020	20983	21136
127	5	6	7	2010332	2010568	18826	18940	1	3	4	388678	393677	21040	21133	1	3	4	391174	396108	21020	21136
131	5	6	7	2010471	2010362	18839	18945	1	3	4	390391	393657	21030	21117	1	3	4	397765	391871	21040	21127
137	5	6	7	2012345	2014684	18861	18941	1	3	4	393321	394077	21122	21119	1	2	7	427820	430894	18996	18969
139	5	6	7	1999306	2007412	18840	18941	1	3	4	391079	392950	20994	21103	1	3	4	395408	392691	21028	21013
149	5	6	7	2002733	1992842	18840	18941	1	3	4	389430	393028	20741	21185	1	3	4	396444	390676	21026	21129
151	5	6	7	2009786	2000141	18840	18940	1	3	4	390824	391927	21123	21192	1	2	7	429203	430614	18869	18959
157	5	6	7	1999286	2001623	18838	18941	1	3	4	392072	391883	21107	21103	1	2	7	426748	428712	18947	19034
163	5	6	7	2003681	2002798	18839	18938	1	3	4	392248	390597	21089	21159	1	2	7	431472	428878	19005	19017
167	5	6	7	2010373	2001412	18838	18939	1	3	4	387801	395073	21124	21085	1	3	4	394994	393623	21003	21132
173	4	5	6	1373038	1321153	17766	17887	1	3	4	390904	391247	21125	21192	1	3	4	403105	407418	20754	21183
179	5	6	7	2002932	1999298	18837	18941	1	3	4	392322	391538	21123	21072	1	3	4	393292	394409	21005	21100
181	5	6	7	2005442	1997862	18834	18934	1	3	4	390675	391940	20959	21084	1	3	4	393198	389579	20893	21116
191	5	6	7	2020260	2083311	18837	18907	1	3	4	390982	399018	21035	21092	1	3	4	392490	390036	20968	21131
193	5	6	7	1988775	1988458	18839	18942	1	3	4	391951	387505	21042	21105	1	2	5	349778	350814	20768	20828
197	5	6	7	2000924	2003578	18838	18942	1	3	4	391142	389521	21087	21086	1	3	4	389648	391615	20888	21098
199	5	6	7	1999201	2002452	18826	18937	1	3	4	392054	396111	21072	21084	1	3	4	392547	391349	21013	21130
211	5	6	7	2000139	2002175	18846	18942	1	3	4	391130	392302	21052	21102	1	2	5	351365	346659	20766	20839
223	5	6	7	2005871	2003351	18826	18941	1	3	4	391778	388674	21073	21100	1	3	4	393320	387469	21002	21099
227	5	6	7	2006891	2008291	18825	18939	1	3	4	388572	392341	21087	21082	1	3	4	392647	393258	21056	21095
229	5	6	7	1999592	1993256	18827	18941	1	3	4	392774	392946	21073	21143	1	2	5	350708	349789	20760	20847
233	5	6	7	2000829	2003679	18825	18942	1	3	4	388306	394773	21073	21083	1	3	4	394638	392937	21037	21141
239	5	6	7	1994224	1998595	18842	18943	1	3	4	392199	390334	21073	21086	1	3	4	392191	395643	20883	21136
241	5	6	7	2000916	2012836	18840	18939	1	3	4	394047	390363	21065	20959	1	2	5	348700	350607	20850	20828
251	5	6	7	2000349	1997253	18825	18938	1	3	4	393759	390761	21086	21199	1	3	4	391785	389722	21011	21119
257	4	5	7	1464328	1455088	17678	17802	1	3	4	391233	390569	20769	21164	1	2	5	350839	350927	20764	20829
263	5	6	7	2009830	2003517	18840	18942	1	3	4	391471	392083	20768	21133	1	2	5	350736	348208	20849	20860
269	5	6	7	2001558	1998064	18837	18935	1	3	4	385184	390676	21075	21085	1	3	4	390305	387963	21000	21104
271	4	5	6	1299158	1300849	17805	17914	1	3	4	388258	388347	21069	21146	1	2	5	350670	349412	20751	20851
277	5	6	7	2005859	2010474	18837	18941	1	3	4	392155	392289	21272	21107	1	2	5	348315	350947	20775	20858
281	5	6	7	1994619	2013941	18838	18942	1	3	4	392830	389383	21220	21160	1	2	5	351300	350792	20769	20840
283	5	6	7	1998433	1995747	18760	18947	1	3	4	388904	390866	20968	21191	1	2	5	351607	350790	20769	20842
293	4	6	7	1303400	1296899	17804	17914	1	3	4	390345	392411	21049	21104	1	2	5	347673	348369	20770	20868
307	5	6	7	2002837	2003962	18836	18941	1	3	4	391735	392271	21038	21161	1	3	4	391307	393105	20998	21089
311	5	6	7	2007660	2016594	18836	18937	1	3	4	391686	392429	20872	21140	1	3	4	393142	392302	21101	21388
313	5	6	7	1995528	2004981	18827	18940	1	3	4	391236	389993	20958	21115	1	2	5	352894	349684	20806	20845

A.2 Segunda batería

El conjunto de combinaciones que se prueban en esta batería son las referenciadas en la tabla 3.4, que se incluye, también, en este apéndice para identificar rápidamente los valores r, s y t de cada combinación. Estas son nombradas por tres o cuatro caracteres que indican de forma orientativa el tamaño de sus bloques según la siguiente tabla:

	Tamaños				+
1		$= 2^0$	1		
S	*small*	$\cong 2^2$	2	3	5
M	*medium*	$\cong 2^4$	13	15	19
L	*large*	$\cong 2^6$	$59^{(*)}$	61	67

$^{(*)}$ 58 si $p = 17$

p	**2**			**5**			**17**			**257**			**65537**		
	r	***s***	***t***	***r***	***s***	***t***	***r***	***s***	***t***	***r***	***s***	***t***	***r***	***s***	***t***
1SS													1	2	7
SSS													2	3	5
1SM										1	2	19	1	2	19
1SM+										1	5	19	1	5	19
SSM										2	3	15	2	3	15
SSM+										3	5	19	3	5	19
1MM							1	15	19	1	15	19	1	15	19
SMM							5	15	19	5	15	19	5	15	19
MMM							13	15	19	13	15	19	13	15	19
1ML				1	19	59	1	19	58	1	19	59			
SML				5	19	59	5	19	58	5	19	59			
MML				15	19	59	15	19	58	15	19	59			
1SL				1	5	59	1	5	58	1	5	59			
SSL				2	5	59	2	5	58	2	5	59			
1SL+				1	2	67	1	2	67						
SSL+				2	3	67	2	3	67						
1LL	1	61	67	1	61	67	1	61	67						
SLL	5	61	67	5	61	67	5	61	67						
MLL	15	61	67	15	61	67	15	61	67						
LLL	59	61	67	59	61	67	58	61	67						

Cada combinación se ha probado con y sin filtrado XOR y aplicando o no la variante *r0* a la extracción (véase la sección 3.3.2), repitiéndose cada prueba 10 veces, cada una con una semilla distinta, y obteniendo en cada caso una secuencia de 200.000 bits.

Los test aplicados son los mismos que se han aplicado a la primera batería, frecuencia (o monobit), serial (o parejas), póker 8 y 16, rachas y autocorrelación; además de obtener la complejidad lineal (véase la sección 1.3). También se repiten los niveles de significación (definición 1.34) $\alpha = 0{,}1$ y $\alpha = 0{,}001$, indicando para cada caso, de las 10 secuencias obtenidas, el número que se ha rechazado de éstas. Las celdas coloreadas indican que se ha producido algún rechazo: anaranjado en cualquier caso salvo para un sólo rechazo para el α más restrictivo (coincide con el 10%), que se usa el amarillo. Para el caso de la complejidad lineal se indica cuántos de los resultados del test tienen una diferencia con la mitad de la secuencia de más de 2 o 4, indicando con el color amarillo cuando es un solo caso y rojo si son más.

Las últimas tablas hacen referencia al coste computacional de las pruebas, medido con el mínimo y la media de ciclos de CPU por iteración y bit extraído, considerando que, dependiendo de la combinación, el *bpi* es distinto (tabla 3.6).

La información de los resultados se encuentra disponible para su tratamiento en el enlace:

http://www.dccia.ua.es/~fmartine/tesis/Bateria2.zip

Universitat d'Alacant
Universidad de Alicante

Test de frecuencia

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					2					0
SSS					4					0
1SM				7	2				0	0
1SM+				8	0				0	0
SSM				7	0				1	0
SSM+				4	0				1	0
1MM			10	4	1			10	0	0
SMM			10	6	1			10	1	0
MMM			10	3	1			10	0	0
1ML		10	10	4			10	10	0	
SML		10	10	6			10	10	0	
MML		10	10	5			10	10	1	
1SL		10	10	4			10	10	1	
SSL		10	10	6			10	10	1	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	2	10	10			0	10	10		
SLL	0	10	10			0	10	10		
MLL	0	10	10			0	10	10		
LLL	0	10	10			0	10	10		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				0	0				0	0
1SM+				1	1				0	0
SSM				1	3				0	0
SSM+				2	1				0	0
1MM			0	0	0			0	0	0
SMM			1	1	3			0	0	0
MMM			1	2	0			0	0	0
1ML		2	1	0			0	0	0	
SML		2	1	1			0	0	0	
MML		1	1	1			0	0	0	
1SL		2	2	0			1	0	0	
SSL		1	2	1			0	0	0	
1SL+		2	2				0	0		
SSL+		4	2				0	0		
1LL	1	4	2			0	0	0		
SLL	0	4	1			0	0	1		
MLL	0	0	2			0	0	0		
LLL	0	1	3			0	0	0		

Con extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					2					0
SSS					4					0
1SM				1	2				0	0
1SM+				0	0				0	0
SSM				2	0				0	0
SSM+				2	0				0	0
1MM			10	1	1			10	0	0
SMM			10	2	1			10	0	0
MMM			10	2	1			10	0	0
1ML		10	10	0			10	10	0	
SML		10	10	0			10	10	0	
MML		10	10	1			10	10	0	
1SL		10	10	3			10	10	0	
SSL		10	10	1			10	10	0	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	N/A	10	10			N/A	10	10		
SLL	N/A	10	10			N/A	10	10		
MLL	N/A	10	10			N/A	10	10		
LLL	N/A	10	10			N/A	10	10		

Con filtrado y extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				0	0				0	0
1SM+				2	1				0	0
SSM				0	3				0	0
SSM+				0	1				0	0
1MM			1	1	0			0	0	0
SMM			1	0	4			0	0	0
MMM			1	0	0			0	0	0
1ML		2	1	0			0	0	0	
SML		1	2	0			0	0	0	
MML		2	1	2			0	0	0	
1SL		2	2	0			0	0	0	
SSL		0	1	2			0	0	0	
1SL+		4	1				1	0		
SSL+		3	1				0	0		
1LL	N/A	3	0			N/A	0	0		
SLL	N/A	1	1			N/A	0	0		
MLL	N/A	0	1			N/A	0	0		
LLL	N/A	1	2			N/A	0	0		

Test serial

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					3					0
1SM				6	0				3	0
1SM+				8	0				1	0
SSM				6	0				3	0
SSM+				7	0				1	0
1MM			10	5	1			10	1	0
SMM			10	5	0			10	1	0
MMM			10	3	0			10	0	0
1ML		10	10	4			10	10	0	
SML		10	10	5			10	10	0	
MML		10	10	5			10	10	2	
1SL		10	10	6			10	10	1	
SSL		10	10	5			10	10	0	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	1	10	10			0	10	10		
SLL	1	10	10			0	10	10		
MLL	1	10	10			0	10	10		
LLL	0	10	10			0	10	10		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					0					0
1SM				1	1				0	0
1SM+				2	1				0	0
SSM				1	0				0	0
SSM+				0	2				0	0
1MM			0	0	0			0	0	0
SMM			1	0	0			0	0	0
MMM			1	0	0			0	0	0
1ML		10	0	0			10	0	0	
SML		10	1	1			10	0	0	
MML		10	2	1			10	0	0	
1SL		10	1	0			10	0	0	
SSL		10	0	1			10	0	0	
1SL+		10	1				10	0		
SSL+		10	0				10	0		
1LL	1	10	2			0	10	0		
SLL	0	10	2			0	10	0		
MLL	0	10	2			0	10	0		
LLL	0	10	1			0	10	0		

Con extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					3					0
1SM				2	0				0	0
1SM+				0	0				0	0
SSM				1	0				0	0
SSM+				0	0				0	0
1MM			10	0	1			10	0	0
SMM			10	1	0			10	0	0
MMM			10	1	0			10	0	0
1ML		10	10	1			10	10	0	
SML		10	10	1			10	10	0	
MML		10	10	1			10	10	0	
1SL		10	10	2			10	10	1	
SSL		10	10	0			10	10	0	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	N/A	10	10			N/A	10	10		
SLL	N/A	10	10			N/A	10	10		
MLL	N/A	10	10			N/A	10	10		
LLL	N/A	10	10			N/A	10	10		

Con filtrado y extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					0					0
1SM				0	1				0	0
1SM+				1	1				0	0
SSM				0	0				0	0
SSM+				0	2				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		10	0	0			10	0	0	
SML		10	1	0			10	0	0	
MML		10	0	1			10	0	0	
1SL		10	4	1			10	0	0	
SSL		10	1	0			10	0	0	
1SL+		10	0				10	0		
SSL+		10	2				10	0		
1LL	N/A	10	1			N/A	10	0		
SLL	N/A	10	0			N/A	10	0		
MLL	N/A	10	0			N/A	10	0		
LLL	N/A	10	3			N/A	10	0		

Universidad de Alicante

Test de póker 8

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					2					0
1SM				10	1				6	0
1SM+				10	0				8	0
SSM				10	0				8	0
SSM+				10	0				7	0
1MM			10	10	0			10	5	0
SMM			10	10	2			10	5	0
MMM			10	10	2			10	6	0
1ML		10	10	10			10	10	7	
SML		10	10	10			10	10	7	
MML		10	10	10			10	10	8	
1SL		10	10	10			10	10	4	
SSL		10	10	10			10	10	8	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	0	10	10			0	10	10		
SLL	1	10	10			0	10	10		
MLL	2	10	10			0	10	10		
LLL	0	10	10			0	10	10		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				1	0				0	0
1SM+				2	4				0	0
SSM				0	1				0	0
SSM+				0	3				0	0
1MM			3	0	0			0	0	0
SMM			2	0	1			0	0	0
MMM			0	0	0			0	0	0
1ML		10	2	1			10	0	0	
SML		10	2	3			10	0	0	
MML		10	3	1			10	0	0	
1SL		10	1	1			10	0	0	
SSL		10	1	3			10	0	0	
1SL+		10	1				10	0		
SSL+		10	3				10	0		
1LL	1	10	2			0	10	0		
SLL	0	10	1			0	10	0		
MLL	1	10	1			0	10	0		
LLL	0	10	5			0	10	0		

Con extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					2					0
1SM				1	1				0	0
1SM+				2	0				0	0
SSM				1	0				0	0
SSM+				2	0				0	0
1MM			10	1	0			10	0	0
SMM			10	0	2			10	0	0
MMM			10	0	2			10	0	0
1ML		10	10	2			10	10	0	
SML		10	10	2			10	10	0	
MML		10	10	0			10	10	0	
1SL		10	10	2			10	10	0	
SSL		10	10	4			10	10	1	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	N/A	10	10			N/A	10	10		
SLL	N/A	10	10			N/A	10	10		
MLL	N/A	10	10			N/A	10	10		
LLL	N/A	10	10			N/A	10	10		

Con filtrado y extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				1	0				0	0
1SM+				2	4				0	0
SSM				1	1				0	0
SSM+				1	3				0	0
1MM			2	1	0			0	0	0
SMM			1	1	1			0	0	0
MMM			0	1	0			0	0	0
1ML		10	1	3			10	0	0	
SML		10	1	0			10	0	0	
MML		10	2	1			10	0	1	
1SL		10	0	1			10	0	0	
SSL		10	1	0			10	0	0	
1SL+		10	0				10	0		
SSL+		10	3				10	0		
1LL	N/A	10	3			N/A	10	0		
SLL	N/A	10	1			N/A	10	0		
MLL	N/A	10	1			N/A	10	0		
LLL	N/A	10	2			N/A	10	0		

Test de póker 16

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					2					1
1SM				1	1				0	0
1SM+				2	1				0	0
SSM				0	1				0	0
SSM+				0	1				0	0
1MM			10	0	2			10	0	0
SMM			10	1	3			10	0	0
MMM			10	1	1			10	0	0
1ML		10	10	2			10	10	0	
SML		10	10	1			10	10	0	
MML		10	10	0			10	10	0	
1SL		10	10	1			10	10	0	
SSL		10	10	4			10	10	0	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	0	10	10			0	10	10		
SLL	0	10	10			0	10	10		
MLL	0	10	10			0	10	10		
LLL	1	10	10			0	10	10		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				2	1				0	0
1SM+				1	0				0	0
SSM				2	1				0	0
SSM+				2	0				0	0
1MM			9	1	2			4	0	0
SMM			10	1	1			6	0	0
MMM			10	2	0			8	0	0
1ML		10	10	1			10	8	0	
SML		10	10	2			10	9	0	
MML		10	10	0			10	9	0	
1SL		10	10	0			10	7	0	
SSL		10	10	1			10	6	0	
1SL+		10	10				10	6		
SSL+		10	9				10	6		
1LL	2	10	9			0	10	7		
SLL	1	10	10			0	10	9		
MLL	0	10	10			0	10	8		
LLL	3	10	7			0	10	5		

Con extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					2					1
1SM				1	1				0	0
1SM+				1	1				0	0
SSM				0	1				0	0
SSM+				0	1				0	0
1MM			5	0	2			2	0	0
SMM			7	0	3			1	0	0
MMM			7	1	1			2	0	0
1ML		10	7	1			10	3	0	
SML		10	9	1			10	1	0	
MML		10	6	0			10	1	0	
1SL		10	8	1			10	1	0	
SSL		10	7	2			10	2	0	
1SL+		10	5				10	2		
SSL+		10	7				10	1		
1LL	N/A	10	7			N/A	10	2		
SLL	N/A	10	7			N/A	10	2		
MLL	N/A	10	8			N/A	10	1		
LLL	N/A	10	6			N/A	10	1		

Con filtrado y extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				0	1				0	0
1SM+				0	0				0	0
SSM				0	1				0	0
SSM+				1	0				0	0
1MM			5	1	2			2	0	0
SMM			3	1	1			0	0	0
MMM			6	0	0			0	0	0
1ML		10	2	1			10	1	0	
SML		10	4	1			10	1	0	
MML		10	3	2			10	1	0	
1SL		10	5	1			10	1	0	
SSL		10	4	1			10	1	0	
1SL+		10	4				10	0		
SSL+		10	6				10	0		
1LL	N/A	10	4			N/A	10	1		
SLL	N/A	10	0			N/A	10	0		
MLL	N/A	10	3			N/A	10	0		
LLL	N/A	10	5			N/A	10	1		

Universidad de Alicante

Test de rachas

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				9	2				7	0
1SM+				10	0				7	0
SSM				9	1				6	0
SSM+				10	0				4	0
1MM			10	9	1			10	5	0
SMM			10	8	2			10	5	0
MMM			10	10	2			10	5	0
1ML		10	10	10			10	10	6	
SML		10	10	9			10	10	8	
MML		10	10	9			10	10	7	
1SL		10	10	10			10	10	2	
SSL		10	10	10			10	10	6	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	1	10	10			0	10	10		
SLL	0	10	10			0	10	10		
MLL	2	10	10			0	10	10		
LLL	0	10	10			0	10	10		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				3	1				1	0
1SM+				1	1				0	0
SSM				2	1				0	0
SSM+				1	0				0	0
1MM			2	1	0			0	0	0
SMM			8	3	0			1	0	0
MMM			2	2	0			0	0	0
1ML		10	4	2			10	1	0	
SML		10	5	3			10	0	0	
MML		10	9	0			10	0	0	
1SL		10	2	2			10	0	0	
SSL		10	6	2			10	0	0	
1SL+		10	4				10	1		
SSL+		10	8				10	2		
1LL	0	10	2			0	10	1		
SLL	0	10	5			0	10	1		
MLL	1	10	7			0	10	1		
LLL	2	10	4			0	10	1		

Con extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				1	2				0	0
1SM+				0	0				0	0
SSM				1	1				0	0
SSM+				0	0				0	0
1MM			10	0	1			10	0	0
SMM			10	1	2			10	0	0
MMM			10	1	2			10	0	0
1ML		10	10	1			10	10	0	
SML		10	10	2			10	10	0	
MML		10	10	1			10	10	0	
1SL		10	10	3			10	10	0	
SSL		10	10	3			10	10	1	
1SL+		10	10				10	10		
SSL+		10	10				10	10		
1LL	N/A	10	10			N/A	10	10		
SLL	N/A	10	10			N/A	10	10		
MLL	N/A	10	10			N/A	10	10		
LLL	N/A	10	10			N/A	10	10		

Con filtrado y extracción *Or*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					1					0
SSS					2					0
1SM				0	1				0	0
1SM+				1	1				0	0
SSM				0	1				0	0
SSM+				0	1				0	0
1MM			3	1	0			0	0	0
SMM			3	1	1			0	0	0
MMM			1	1	0			0	0	0
1ML		10	5	0			10	1	0	
SML		10	1	1			10	0	0	
MML		10	2	1			10	0	0	
1SL		10	3	1			10	0	0	
SSL		10	2	1			10	0	0	
1SL+		10	2				10	0		
SSL+		10	5				10	0		
1LL	N/A	10	2			N/A	10	1		
SLL	N/A	10	6			N/A	10	0		
MLL	N/A	10	1			N/A	10	0		
LLL	N/A	10	5			N/A	10	0		

Test de autocorrelación

Sin filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	0				0	0
1MM			10	0	0			0	0	0
SMM			7	0	0			0	0	0
MMM			10	0	0			0	0	0
1ML		10	8	0			10	0	0	
SML		10	9	0			10	0	0	
MML		10	10	0			10	0	0	
1SL		10	10	0			10	0	0	
SSL		10	9	0			10	0	0	
1SL+		10	8				10	0		
SSL+		10	10				10	0		
1LL	0	10	10			0	10	0		
SLL	0	10	9			0	10	0		
MLL	0	10	9			0	10	0		
LLL	0	10	9			0	10	0		

Con filtrado

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	0				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		0	0	0			0	0	0	
SML		0	0	0			0	0	0	
MML		0	0	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	0				0	0		
1LL	0	0	0			0	0	0		
SLL	0	0	0			0	0	0		
MLL	0	0	0			0	0	0		
LLL	0	0	0			0	0	0		

Con extracción *0r*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	0				0	0
1MM			1	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			1	0	0			0	0	0
1ML		10	1	0			10	0	0	
SML		10	1	0			10	0	0	
MML		10	0	0			10	0	0	
1SL		10	1	0			10	0	0	
SSL		10	1	0			10	0	0	
1SL+		10	1				10	0		
SSL+		10	0				10	0		
1LL	N/A	10	1			N/A	10	0		
SLL	N/A	10	0			N/A	10	0		
MLL	N/A	10	0			N/A	10	0		
LLL	N/A	10	0			N/A	10	0		

Con filtrado y extracción *0r*

	α = 0,1					α = 0,001				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	0				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		0	0	0			0	0	0	
SML		0	0	0			0	0	0	
MML		0	0	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	0				0	0		
1LL	N/A	0	0			N/A	0	0		
SLL	N/A	0	0			N/A	0	0		
MLL	N/A	0	0			N/A	0	0		
LLL	N/A	0	0			N/A	0	0		

Universidad de Alicante

Complejidad lineal

Sin filtrado

	(100000-LC) > 2					(100000-LC) > 4				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	1				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		1	0	0			0	0	0	
SML		0	0	1			0	0	0	
MML		0	0	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	2				0	0		
1LL	0	0	0			0	0	0		
SLL	0	0	0			0	0	0		
MLL	0	0	1			0	0	0		
LLL	0	0	0			0	0	0		

Con filtrado

	(100000-LC) > 2					(100000-LC) > 4				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					1					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	1				0	0
SSM+				0	0				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		0	0	0			0	0	0	
SML		0	0	0			0	0	0	
MML		0	1	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	0				0	0		
1LL	0	0	0			0	0	0		
SLL	0	1	0			0	0	0		
MLL	0	0	0			0	0	0		
LLL	0	0	0			0	0	0		

Con extracción *Or*

	(100000-LC) > 2					(100000-LC) > 4				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					0					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	0				0	0
SSM+				0	0				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		0	1	0			0	0	0	
SML		0	0	0			0	0	0	
MML		0	0	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	0				0	0		
1LL	N/A	0	1			N/A	0	0		
SLL	N/A	0	0			N/A	0	0		
MLL	N/A	0	0			N/A	0	0		
LLL	N/A	0	0			N/A	0	0		

Con filtrado y extracción *Or*

	(100000-LC) > 2					(100000-LC) > 4				
	2	5	17	257	65537	2	5	17	257	65537
1SS					0					0
SSS					1					0
1SM				0	0				0	0
1SM+				0	0				0	0
SSM				0	1				0	0
SSM+				0	0				0	0
1MM			0	0	0			0	0	0
SMM			0	0	0			0	0	0
MMM			0	0	0			0	0	0
1ML		0	0	0			0	0	0	
SML		0	0	0			0	0	0	
MML		0	0	0			0	0	0	
1SL		0	0	0			0	0	0	
SSL		0	0	0			0	0	0	
1SL+		0	0				0	0		
SSL+		0	0				0	0		
1LL	N/A	0	0			N/A	0	0		
SLL	N/A	1	0			N/A	0	0		
MLL	N/A	0	0			N/A	0	0		
LLL	N/A	0	0			N/A	0	0		

Mínimo de ciclos de CPU por iteración

Sin filtrado

	2	5	17	257	65537
1SS					12
SSS					14
1SM				42	54
1SM+				82	84
SSM				53	55
SSM+				133	162
1MM			381	380	383
SMM			515	552	551
MMM			891	884	890
1ML		2519	2356	2558	
SML		2992	2946	3060	
MML		5011	4488	4946	
1SL		613	558	592	
SSL		703	641	661	
1SL+		378	347		
SSL+		610	604		
1LL	18165	18794	18235		
SLL	19349	19336	19435		
MLL	23047	23953	22991		
LLL	50107	51933	49345		

Con filtrado

	2	5	17	257	65537
1SS					13
SSS					14
1SM				42	56
1SM+				82	83
SSM				54	56
SSM+				134	163
1MM			380	378	386
SMM			515	552	569
MMM			894	885	889
1ML		2520	2356	2557	
SML		2992	2937	3063	
MML		5019	4492	4935	
1SL		614	558	592	
SSL		705	641	662	
1SL+		379	347		
SSL+		611	605		
1LL	17328	18780	18203		
SLL	18455	19320	19377		
MLL	21972	23918	22939		
LLL	47747	51776	49197		

Con extracción *Or*

	2	5	17	257	65537
1SS					12
SSS					15
1SM				43	55
1SM+				83	84
SSM				55	56
SSM+				136	163
1MM			387	383	389
SMM			521	557	572
MMM			900	893	894
1ML		2542	2371	2571	
SML		3023	2958	3077	
MML		5062	4519	4955	
1SL		620	563	595	
SSL		714	646	666	
1SL+		382	350		
SSL+		617	608		
1LL	N/A	18866	18260		
SLL	N/A	19423	19442		
MLL	N/A	24042	23022		
LLL	N/A	51990	49351		

Con filtrado y extracción *Or*

	2	5	17	257	65537
1SS					12
SSS					15
1SM				43	55
1SM+				83	85
SSM				55	56
SSM+				136	164
1MM			384	381	387
SMM			523	556	576
MMM			908	900	895
1ML		2546	2372	2572	
SML		3026	2961	3095	
MML		5074	4522	4967	
1SL		622	562	596	
SSL		714	646	666	
1SL+		383	350		
SSL+		618	609		
1LL	N/A	18871	18263		
SLL	N/A	19433	19443		
MLL	N/A	24061	23027		
LLL	N/A	52034	49359		

Universitat d'Alacant
Universidad de Alicante

Mínimo de ciclos de CPU por bit

Sin filtrado

	2	5	17	257	65537
1SS					0,033
SSS					0,028
1SM				0,089	0,057
1SM+				0,086	0,044
SSM				0,082	0,042
SSM+				0,1	0,061
1MM			0,299	0,149	0,075
SMM			0,283	0,152	0,076
MMM			0,306	0,152	0,077
1ML		1,05	0,5	0,267	
SML		0,99	0,495	0,253	
MML		1,094	0,497	0,27	
1SL		0,854	0,395	0,206	
SSL		0,831	0,385	0,195	
1SL+		0,931	0,427		
SSL+		0,894	0,443		
1LL	4,31	2,229	1,082		
SLL	4,093	2,045	1,028		
MLL	3,837	1,994	0,957		
LLL	4,305	2,231	1,072		

Con filtrado

	2	5	17	257	65537
1SS					0,035
SSS					0,028
1SM				0,089	0,059
1SM+				0,086	0,044
SSM				0,083	0,043
SSM+				0,1	0,061
1MM			0,298	0,148	0,076
SMM			0,283	0,152	0,078
MMM			0,307	0,152	0,076
1ML		1,051	0,5	0,267	
SML		0,99	0,494	0,253	
MML		1,095	0,498	0,269	
1SL		0,855	0,395	0,206	
SSL		0,833	0,385	0,196	
1SL+		0,933	0,427		
SSL+		0,896	0,444		
1LL	4,111	2,228	1,08		
SLL	3,904	2,044	1,025		
MLL	3,658	1,991	0,955		
LLL	4,102	2,224	1,068		

Con extracción *Or*

	2	5	17	257	65537
1SS					0,033
SSS					0,03
1SM				0,091	0,058
1SM+				0,087	0,044
SSM				0,085	0,043
SSM+				0,102	0,061
1MM			0,303	0,15	0,076
SMM			0,286	0,153	0,079
MMM			0,309	0,154	0,077
1ML		1,06	0,503	0,268	
SML		1	0,497	0,255	
MML		1,105	0,501	0,27	
1SL		0,864	0,399	0,207	
SSL		0,844	0,388	0,197	
1SL+		0,941	0,431		
SSL+		0,905	0,446		
1LL	N/A	2,238	1,083		
SLL	N/A	2,054	1,028		
MLL	N/A	2,001	0,958		
LLL	N/A	2,233	1,072		

Con filtrado y extracción *Or*

	2	5	17	257	65537
1SS					0,033
SSS					0,03
1SM				0,091	0,058
1SM+				0,087	0,045
SSM				0,085	0,043
SSM+				0,102	0,061
1MM			0,301	0,149	0,076
SMM			0,287	0,153	0,079
MMM			0,312	0,155	0,077
1ML		1,062	0,503	0,268	
SML		1,001	0,498	0,256	
MML		1,107	0,501	0,271	
1SL		0,866	0,398	0,208	
SSL		0,844	0,388	0,197	
1SL+		0,943	0,431		
SSL+		0,906	0,446		
1LL	N/A	2,239	1,083		
SLL	N/A	2,056	1,028		
MLL	N/A	2,003	0,958		
LLL	N/A	2,235	1,072		

Universitat d'Alacant
Universidad de Alicante

Media de ciclos de CPU por iteración

Sin filtrado

	2	5	17	257	65537
1SS					18,5
SSS					24,2
1SM				54,4	62,2
1SM+				124,5	130,3
SSM				74,9	79,8
SSM+				168,5	192,4
1MM			402,6	474,8	490,2
SMM			583,4	580,3	672,2
MMM			975,3	991,3	1132
1ML		2871	2697	2747	
SML		3492	3105	3446	
MML		5346	4902	5759	
1SL		744,8	578,1	649,8	
SSL		850,6	689,4	771,6	
1SL+		420,9	380,8		
SSL+		725,7	680,5		
1LL	18813	20184	18704		
SLL	20184	21262	20306		
MLL	24191	25109	23819		
LLL	52756	54069	50671		

Con filtrado

	2	5	17	257	65537
1SS					27,6
SSS					35
1SM				69,7	81,8
1SM+				154,4	173,4
SSM				84,2	117,1
SSM+				179,6	256
1MM			499,5	529,5	724,1
SMM			661,5	794,2	925,1
MMM			1173	1086	1463
1ML		3582	3126	3185	
SML		4657	3874	3946	
MML		6169	6006	6515	
1SL		793,2	692,8	840,1	
SSL		932,9	793,9	914,8	
1SL+		541,1	421,7		
SSL+		781,7	860,4		
1LL	19217	21272	21600		
SLL	19480	23858	26008		
MLL	23352	27635	27563		
LLL	50503	59043	59099		

Con extracción *Or*

	2	5	17	257	65537
1SS					21,6
SSS					28,8
1SM				71,9	86,1
1SM+				148,6	161,5
SSM				83	92
SSM+				219	238
1MM			506,1	537,1	615,2
SMM			714,2	816,8	914,1
MMM			1156	1434	1453
1ML		3067	3006	3178	
SML		3793	3911	4118	
MML		6329	5717	6478	
1SL		837,4	823,3	736	
SSL		939,3	829,4	958,5	
1SL+		512	459,2		
SSL+		753,6	762,3		
1LL	N/A	22569	21850		
SLL	N/A	23052	24204		
MLL	N/A	28889	26130		
LLL	N/A	58394	53538		

Con filtrado y extracción *Or*

	2	5	17	257	65537
1SS					23,1
SSS					31,7
1SM				70,6	79,6
1SM+				139,6	179,7
SSM				92	103,7
SSM+				189,2	248
1MM			518	565	592
SMM			745	823,4	1007
MMM			1400	1310	1455
1ML		3802	2982	3233	
SML		4274	3390	4759	
MML		6085	6022	7774	
1SL		852,6	734,2	891,5	
SSL		981,9	848,5	1014	
1SL+		507,1	422,5		
SSL+		815,4	771,9		
1LL	N/A	21084	22161		
SLL	N/A	23127	22335		
MLL	N/A	29414	27820		
LLL	N/A	57353	57305		

Universitat d'Alacant
Universidad de Alicante

Media de ciclos de CPU por bit

Sin filtrado

	2	5	17	257	65537
1SS					0,05
SSS					0,049
1SM				0,115	0,066
1SM+				0,131	0,068
SSM				0,116	0,062
SSM+				0,126	0,072
1MM			0,316	0,186	0,096
SMM			0,321	0,159	0,092
MMM			0,335	0,17	0,097
1ML		1,197	0,572	0,286	
SML		1,155	0,522	0,285	
MML		1,167	0,543	0,314	
1SL		1,037	0,409	0,226	
SSL		1,005	0,414	0,228	
1SL+		1,037	0,469		
SSL+		1,064	0,499		
1LL	4,463	2,394	1,109		
SLL	4,27	2,249	1,074		
MLL	4,027	2,09	0,991		
LLL	4,533	2,323	1,1		

Con filtrado

	2	5	17	257	65537
1SS					0,075
SSS					0,071
1SM				0,148	0,087
1SM+				0,162	0,091
SSM				0,13	0,09
SSM+				0,134	0,096
1MM			0,391	0,207	0,142
SMM			0,363	0,218	0,127
MMM			0,403	0,187	0,126
1ML		1,494	0,663	0,332	
SML		1,541	0,651	0,326	
MML		1,346	0,665	0,355	
1SL		1,105	0,491	0,293	
SSL		1,103	0,477	0,27	
1SL+		1,333	0,519		
SSL+		1,146	0,631		
1LL	4,559	2,523	1,281		
SLL	4,121	2,524	1,376		
MLL	3,888	2,3	1,147		
LLL	4,339	2,536	1,284		

Con extracción *Or*

	2	5	17	257	65537
1SS					0,059
SSS					0,058
1SM				0,152	0,091
1SM+				0,156	0,085
SSM				0,128	0,071
SSM+				0,164	0,089
1MM			0,397	0,21	0,121
SMM			0,392	0,224	0,126
MMM			0,397	0,247	0,125
1ML		1,279	0,637	0,331	
SML		1,255	0,658	0,341	
MML		1,381	0,633	0,353	
1SL		1,166	0,583	0,256	
SSL		1,11	0,498	0,283	
1SL+		1,261	0,566		
SSL+		1,105	0,559		
1LL	N/A	2,677	1,296		
SLL	N/A	2,438	1,28		
MLL	N/A	2,405	1,087		
LLL	N/A	2,509	1,163		

Con filtrado y extracción *Or*

	2	5	17	257	65537
1SS					0,063
SSS					0,064
1SM				0,15	0,084
1SM+				0,147	0,094
SSM				0,142	0,08
SSM+				0,142	0,093
1MM			0,406	0,221	0,116
SMM			0,409	0,226	0,138
MMM			0,482	0,225	0,125
1ML		1,586	0,632	0,337	
SML		1,414	0,57	0,394	
MML		1,328	0,667	0,424	
1SL		1,187	0,52	0,31	
SSL		1,161	0,51	0,3	
1SL+		1,249	0,52		
SSL+		1,196	0,566		
1LL	N/A	2,501	1,314		
SLL	N/A	2,446	1,181		
MLL	N/A	2,448	1,158		
LLL	N/A	2,464	1,245		

Universitat d'Alacant
Universidad de Alicante

A.3 Tercera batería

Esta tercera batería de pruebas se ha realizado con las 9 combinaciones para el primo 65537 que se referencian en la tabla 3.8 que, también, se muestran a continuación para identificar rápidamente los valores r, s y t de cada una de las combinaciones, que son nombradas como en la batería anterior (véase el apéndice A.2):

	r	*s*	*t*	*bpi*	*mbp*	*mbs*
1SS	1	2	7	368	128	368
SSS	2	3	5	496	128	496
1SM	1	2	19	944	320	944
1SM+	1	5	19	1904	368	1904
SSM	2	3	15	1296	256	1296
SSM+	3	5	19	2672	400	2672
1MM	1	15	19	5104	528	5104
SMM	5	15	19	7280	528	7280
MMM	13	15	19	11632	720	11632

Se trata de combinaciones admisibles con las que se han generado y analizado 100 secuencias de 200.000 bits por combinación, con y sin filtrado *s-box* 32×32.

Se aplican los mismos test que en las dos baterías anteriores (frecuencia, serial, póker 8 y 16, rachas y autocorrelación; además de complejidad lineal) con los mismos niveles de significación $\alpha = 0{,}1$ y $\alpha = 0{,}001$ (véase la sección 1.2.2.1). En las celdas se indica el porcentaje de rechazos. Si ha habido algún rechazo se colorea la celda de amarillo y de anaranjado si el porcentaje supera el nivel de significación. Para la complejidad lineal se muestra, de los 100 resultados del test, cuántos difieren respecto a la mitad de la longitud de la secuencia en más de 2 o 4, marcando con color de fondo rojo las celdas donde se da alguno de estos casos. En todos los casos se muestra, además, la media de los resultados obtenidos y su desviación típica.

Las últimas tablas hacen referencia al coste computacional de las pruebas, medido con el mínimo y la media de ciclos de CPU por iteración y por bit extraído, considerando que, dependiendo de la combinación, el *bpi* es distinto.

La información de los resultados se encuentra disponible para su tratamiento en el enlace:

http://www.dccia.ua.es/~fmartine/tesis/Bateria3.zip

Test de frecuencia

	2,706	10,83		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	12%	1%	1,085308	1,77797369
SSS	15%	1%	1,233596	1,97691575
1SM	11%	1%	1,270143	1,90459192
1SM+	8%	0%	0,907082	1,54946367
SSM	8%	0%	0,777172	1,2042298
SSM+	10%	0%	0,944065	1,50980045
1MM	14%	0%	1,230357	1,63135435
SMM	11%	0%	1,032734	1,36636117
MMM	8%	0%	0,933444	1,32136906
	2,706	10,83		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	8%	0%	0,972892	1,38919374
SSS	17%	0%	1,21633	1,71267538
1SM	9%	0%	0,833283	1,10157441
1SM+	4%	0%	0,775118	1,0980578
SSM	8%	0%	0,990002	1,58889093
SSM+	8%	0%	0,998676	1,23024883
1MM	8%	0%	0,806809	1,15109554
SMM	6%	0%	0,79316	1,25462264
MMM	10%	0%	1,05296	1,33674378

Test de póker 8

	284,3	330,5		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	11%	0%	254,4612	23,590824
SSS	10%	0%	256,1031	20,864384
1SM	7%	0%	252,6184	22,425442
1SM+	9%	0%	253,5005	22,384779
SSM	11%	0%	255,8422	23,230507
SSM+	9%	0%	252,5793	21,981336
1MM	8%	0%	253,1668	21,302382
SMM	10%	0%	255,8868	21,141646
MMM	12%	0%	250,5643	23,462795
	284,3	330,5		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	6%	1%	254,0723	21,795865
SSS	11%	0%	255,0395	21,805054
1SM	10%	0%	256,0418	24,456533
1SM+	7%	0%	252,6041	20,820658
SSM	10%	0%	251,8344	25,021035
SSM+	15%	1%	259,2031	23,240819
1MM	11%	0%	257,2932	22,164209
SMM	12%	0%	259,5224	22,499980
MMM	15%	1%	257,8062	23,763354

Test serial

	4,605	13,82		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	6%	1%	1,451128	2,1747082
SSS	7%	1%	1,164423	2,27105071
1SM	9%	0%	1,443906	2,54915227
1SM+	5%	0%	0,957055	1,81259365
SSM	6%	0%	1,050504	1,7172073
SSM+	3%	0%	0,724881	1,72450404
1MM	9%	1%	1,528761	2,5362279
SMM	7%	0%	1,047861	1,81664835
MMM	4%	0%	0,91946	1,90784127
	4,605	13,82		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	5%	0%	1,022985	1,80648799
SSS	12%	0%	1,440153	2,18865206
1SM	3%	0%	0,812414	1,54660064
1SM+	5%	0%	0,963191	1,8032396
SSM	5%	0%	0,951789	2,19668446
SSM+	2%	0%	1,053625	1,55048307
1MM	7%	0%	1,066062	1,95107844
SMM	5%	0%	1,03023	2,00853508
MMM	9%	0%	1,409772	2,09720659

Test de póker 16

	65999,39	66659,46		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	6%	0%	65532,40	317,0150
SSS	11%	0%	65529,36	400,2285
1SM	12%	0%	65497,38	376,6338
1SM+	10%	1%	65541,31	353,2516
SSM	5%	0%	65551,06	355,2281
SSM+	10%	0%	65554	330,521
1MM	4%	0%	65483,22	345,7260
SMM	9%	0%	65533,55	372,0256
MMM	8%	0%	65521,91	339,9976
	65999,39	66659,46		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	7%	0%	65465,61	358,8382
SSS	11%	2%	65556,52	390,8565
1SM	10%	0%	65534,71	366,8137
1SM+	11%	0%	65552,43	323,4924
SSM	7%	0%	65460,26	338,4597
SSM+	11%	0%	65536,80	354,4225
1MM	13%	0%	65534,50	379,0623
SMM	11%	0%	65542,05	335,0542
MMM	15%	0%	65563,65	430,846

Test de rachas

	33,1962	51,1786		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	13%	0%	25,61448	6,8649549
SSS	7%	0%	23,63465	6,0431420
1SM	11%	0%	24,55741	7,3096190
1SM+	10%	0%	23,78081	6,6748841
SSM	12%	0%	25,78328	6,8257657
SSM+	11%	0%	22,70049	7,0168312
1MM	18%	0%	25,57294	7,8210249
SMM	14%	0%	24,85415	7,7563162
MMM	10%	0%	23,78028	6,7048822
	33,1962	51,1786		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	16%	0%	23,81486	8,1198852
SSS	13%	0%	25,14112	7,5493796
1SM	14%	0%	24,95160	7,4624272
1SM+	14%	0%	24,13194	7,1351341
SSM	17%	1%	24,60494	8,0927023
SSM+	11%	0%	24,25231	6,9513500
1MM	11%	1%	25,14703	7,0070605
SMM	10%	0%	24,00491	7,3361986
MMM	12%	0%	23,87469	7,4818458

Test de autocorrelación

	1,282	3,09		
Con *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	0%	0%	0,79771	0,0024932
SSS	0%	0%	0,798317	0,00250386
1SM	0%	0%	0,797857	0,00259947
1SM+	0%	0%	0,798045	0,0026364
SSM	0%	0%	0,797923	0,00277565
SSM+	0%	0%	0,798316	0,00246526
1MM	0%	0%	0,798127	0,00253253
SMM	0%	0%	0,798118	0,00250782
MMM	0%	0%	0,797584	0,00252597
	1,282	3,09		
Sin *s-box*	**α = 10%**	**α = 0,1%**	**Media**	**Desviación**
1SS	0%	0%	0,797884	0,00236283
SSS	0%	0%	0,797607	0,0022644
1SM	0%	0%	0,79781	0,0023603
1SM+	0%	0%	0,797996	0,00228703
SSM	0%	0%	0,797502	0,00253489
SSM+	0%	0%	0,797954	0,00223374
1MM	0%	0%	0,798068	0,00244804
SMM	0%	0%	0,798219	0,00252185
MMM	0%	0%	0,798262	0,00233436

Complejidad lineal

	100000	- LC >		
Con *s-box*	**2**	**4**	**Media**	**Desviación**
1SS	1	0	100000,26	0,96000842
SSS	0	0	100000,25	0,82112266
1SM	0	0	100000,16	0,93980441
1SM+	1	0	100000,05	0,94681817
SSM	1	0	100000,2	0,94280904
SSM+	0	0	100000,37	1,05078134
1MM	1	0	100000,23	1,02351148
SMM	1	0	100000,22	0,92747077
MMM	0	0	100000,13	0,87218523
	100000	- LC >		
Sin *s-box*	**2**	**4**	**Media**	**Desviación**
1SS	4	0	99999,93	1,19134081
SSS	0	0	100000,21	0,93522536
1SM	0	0	100000,13	0,89504811
1SM+	0	0	100000,26	1,00121139
SSM	1	1	100000,2	1,19764078
SSM+	3	0	100000,34	1,30438267
1MM	1	0	100000,09	1,00599215
SMM	0	0	100000,23	0,85108368
MMM	1	0	100000,29	0,94596431

Ciclos de CPU

	Mínimo de ciclos de CPU				**Media de ciclos de CPU**			
Con *s-box*	**E(*)**	**F(*)**	**T(*)**	**Por bit**	**E(*)**	**F(*)**	**T(*)**	**Por bit**
1SS	4	3	7	0,01902	4	5	9	0,02571
SSS	5	5	10	0,02016	6	8	14	0,02752
1SM	18	9	27	0,02860	21	15	36	0,03787
1SM+	43	19	62	0,03256	49	34	83	0,04356
SSM	23	13	36	0,02778	27	35	62	0,04809
SSM+	61	27	88	0,03293	67	46	113	0,04230
1MM	201	52	253	0,04957	222	112	334	0,06547
SMM	267	75	342	0,04698	296	160	456	0,06258
MMM	468	121	589	0,05064	526	277	803	0,06907

	Mínimo de ciclos de CPU				**Media de ciclos de CPU**			
Sin *s-box*	**E(*)**	**F(*)**	**T(*)**	**Por bit**	**E(*)**	**F(*)**	**T(*)**	**Por bit**
1SS	4	3	7	0,01902	4	5	9	0,02508
SSS	5	5	10	0,02016	7	8	14	0,02911
1SM	19	9	28	0,02966	21	14	35	0,03719
1SM+	45	19	64	0,03361	48	31	79	0,04164
SSM	24	13	37	0,02855	27	170	197	0,15228
SSM+	64	27	91	0,03406	70	49	119	0,04443
1MM	211	52	263	0,05153	221	124	345	0,06751
SMM	281	75	356	0,04890	292	141	433	0,05946
MMM	493	119	612	0,05261	509	1037	1546	0,13287

(*) E=Ciclos en cálculos de nueva iteración y extracción
F=Ciclos en filtrado T=Ciclos en total

A.4 Cuarta batería

Esta cuarta batería se refiere a los análisis de test de aleatoriedad PractRand y TestU01 [41, 58, 59], así como a las mediciones de tiempo de CPU para distintas combinaciones con $p = 65537$, $r = 1$ y $s = 2$; que son, en definitiva, las combinaciones del generador propuesto y su familia ($t = d$).

El análisis PractRand (versión 0.92) consiste en un conjunto de test que se van realizando (repitiendo y ampliando) hasta que la secuencia analizada alcance una longitud máxima, o el análisis sea interrumpido por un informe desfavorable. El análisis se puede realizar mostrando todos los resultados de los test (modo *display all*) o de forma que sólo informe cada cierta cantidad de bytes analizados (por defecto, cada vez que se duplica la longitud de la secuencia analizada y empezando con 64MB). Los informes que no interrumpen el análisis pueden ser totalmente favorables o indicar sospechas, que se consideran falsas si posteriormente, analizando más bytes, no se interrumpe el análisis con un informe desfavorable. Las secuencias se han probado hasta alcanzar el Terabyte (290 test por secuencia) a no ser que un informe desfavorable las haya interrumpido, como es el caso (únicamente) de las secuencias generadas sin *s-box*. Se han realizado análisis PractRand a las secuencias generadas sin *s-box* de las combinaciones $p = 65537$, $r = 1$, $s = 2$, y $t = 7, 13, 15$ y 19, y a las generadas con *s-box* 32×32 de $p = 65537$, $r = 1$, $s = 2$, y $t = d$ con $d = 7, 8, \dots, 15$ y 19. Los resultados que se muestran aquí son, por abreviar, sólo con informes puntuales (a los 128MB, 256MB, … y 1TB); los resultados con los informes del modo *display all* se proporcionan en el enlace:

http://www.dccia.ua.es/~fmartine/tesis/Bateria4.zip

Universitat d'Alacant
Universidad de Alicante

Para el análisis TestU01 se pueden pedir cinco tipos de conjuntos de test: alphabit, rabbit, small crush, crush (medium) y big crush. El más estricto y exhaustivo es big crush. Se han realizado análisis con los cinco conjuntos de test (17, 33, 15, 144 y 160 test respectivamente) y con diez claves distintas, a los generadores M7, M10 y M19.

De entre las 150 pruebas, algún test ha proporcionado puntualmente algún valor inusual, sin ningún patrón que pudiera determinar que hay alguna deficiencia en el generador relacionada con un test en concreto. Para los tres generadores hay varias claves que pasan los cinco análisis completos superando satisfactoriamente todos los test.

La longitud de la secuencia no se selecciona previamente, dejando el generador en modo secuencia infinita hasta que la finalización del test lo interrumpa; de este modo la longitud final depende de la necesidad que tenga el conjunto de test solicitado.

Los informes generados por los análisis son extensos, sobre todo en el caso del conjunto big crush (160 test), por lo que aquí sólo se muestra, como ejemplo, uno de los informes generados al analizar el generador M10 con big crush. También se incluye, como prueba de la conveniencia del filtro x-box, un ejemplo (solamente el sumario) de un resultado desfavorable del big crush (a una secuencia obtenida con un generador combinación 65537 1-2-10 sin filtro *s-box*). El resto de informes están disponibles también en el enlace:

http://www.dccia.ua.es/~fmartine/tesis/Bateria4.zip

Una última tabla muestra las velocidades del generador optimizado para los distintos valores de *d* probados. Para poder establecer los tiempos, se han medido ciclos de CPU en la generación de secuencias de un gigabyte, sin escritura al sistema de ficheros, y se ha calculado su equivalencia en segundos. Como comparativa, se facilita la tabla de velocidades de otros generadores calculadas con las mismas condiciones.

Universitat d'Alacant
Universidad de Alicante

Test PractRand con s-box $p = 65537, r = 1, s = 2, t = d$

ConSBox_d7.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0xfb938825
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0xfb938825
length= 64 megabytes (2^26 bytes), time= 2.1 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 128 megabytes (2^27 bytes), time= 5.1 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 256 megabytes (2^28 bytes), time= 10.3 seconds
  no anomalies in 162 test result(s)


rng=RNG_stdin, seed=0xfb938825
length= 512 megabytes (2^29 bytes), time= 19.3 seconds
  no anomalies in 171 test result(s)


rng=RNG_stdin, seed=0xfb938825
length= 1 gigabyte (2^30 bytes), time= 36.9 seconds
  Test Name                         Raw       Processed     Evaluation
  DC6-9x1Bytes-1                    R=  +5.5  p =  5.4e-3   unusual
  [Low1/8]BCFN(2+0,13-3,T)          R=  -7.4  p =1-4.0e-4   unusual
  ...and 181 test result(s) without anomalies

rng=RNG_stdin, seed=0xfb938825
length= 2 gigabytes (2^31 bytes), time= 71.5 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 4 gigabytes (2^32 bytes), time= 153 seconds
  Test Name                         Raw       Processed     Evaluation
  Gap-16:B                          R=  -4.4  p =1-1.2e-3   unusual
  ...and 202 test result(s) without anomalies

rng=RNG_stdin, seed=0xfb938825
length= 8 gigabytes (2^33 bytes), time= 348 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 16 gigabytes (2^34 bytes), time= 739 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]DC6-9x1Bytes-1            R=  +5.4  p =  5.4e-3   unusual
  ...and 225 test result(s) without anomalies

rng=RNG_stdin, seed=0xfb938825
length= 32 gigabytes (2^35 bytes), time= 1524 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]DC6-9x1Bytes-1            R=  +5.5  p =  5.1e-3   unusual
  ...and 234 test result(s) without anomalies

rng=RNG_stdin, seed=0xfb938825
length= 64 gigabytes (2^36 bytes), time= 24764 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 128 gigabytes (2^37 bytes), time= 31500 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 256 gigabytes (2^38 bytes), time= 44561 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 512 gigabytes (2^39 bytes), time= 69159 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0xfb938825
length= 1 terabyte (2^40 bytes), time= 92848 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d8.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x84ef7716
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x84ef7716
length= 64 megabytes (2^26 bytes), time= 2.7 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 128 megabytes (2^27 bytes), time= 7.8 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 256 megabytes (2^28 bytes), time= 15.6 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 512 megabytes (2^29 bytes), time= 30.1 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 1 gigabyte (2^30 bytes), time= 56.6 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 2 gigabytes (2^31 bytes), time= 108 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 4 gigabytes (2^32 bytes), time= 205 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 8 gigabytes (2^33 bytes), time= 404 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 16 gigabytes (2^34 bytes), time= 790 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 32 gigabytes (2^35 bytes), time= 1527 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 64 gigabytes (2^36 bytes), time= 22994 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 128 gigabytes (2^37 bytes), time= 26058 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 256 gigabytes (2^38 bytes), time= 32005 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 512 gigabytes (2^39 bytes), time= 44342 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0x84ef7716
length= 1 terabyte (2^40 bytes), time= 68099 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d9.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0xf82052f4
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0xf82052f4
length= 64 megabytes (2^26 bytes), time= 3.1 seconds
  Test Name                         Raw       Processed     Evaluation
  Gap-16:A                          R=  +4.8  p =  1.4e-3   unusual
  ...and 138 test result(s) without anomalies

rng=RNG_stdin, seed=0xf82052f4
length= 128 megabytes (2^27 bytes), time= 8.4 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]BCFN(2+1,13-5,T)         R=  +8.6  p =  1.5e-3   unusual
  [Low1/32]BCFN(2+0,13-6,T)         R=  -5.8  p =1-7.9e-4   unusual
  ...and 149 test result(s) without anomalies

rng=RNG_stdin, seed=0xf82052f4
length= 256 megabytes (2^28 bytes), time= 16.6 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]Gap-16:A                  R=  +4.7  p =  2.2e-3   unusual
  ...and 161 test result(s) without anomalies

rng=RNG_stdin, seed=0xf82052f4
length= 512 megabytes (2^29 bytes), time= 30.4 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 1 gigabyte (2^30 bytes), time= 61.2 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 2 gigabytes (2^31 bytes), time= 123 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 4 gigabytes (2^32 bytes), time= 231 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 8 gigabytes (2^33 bytes), time= 400 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 16 gigabytes (2^34 bytes), time= 735 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 32 gigabytes (2^35 bytes), time= 1381 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 64 gigabytes (2^36 bytes), time= 22710 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 128 gigabytes (2^37 bytes), time= 25393 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 256 gigabytes (2^38 bytes), time= 30535 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/32]DC6-9x1Bytes-1           R=  -4.4  p =1-6.1e-3   unusual
  ...and 266 test result(s) without anomalies

rng=RNG_stdin, seed=0xf82052f4
length= 512 gigabytes (2^39 bytes), time= 41243 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0xf82052f4
length= 1 terabyte (2^40 bytes), time= 62561 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]Gap-16:B                  R=  -4.5  p =1-9.5e-4   unusual
  ...and 289 test result(s) without anomalies
```

ConSBox_d10.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0xe40f5ee7
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0xe40f5ee7
length= 64 megabytes (2^26 bytes), time= 1.4 seconds
  Test Name                         Raw       Processed     Evaluation
  BCFN(2+1,13-3,T)                  R=  +8.2  p =  1.1e-3   unusual
  ...and 138 test result(s) without anomalies

rng=RNG_stdin, seed=0xe40f5ee7
length= 128 megabytes (2^27 bytes), time= 3.0 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 256 megabytes (2^28 bytes), time= 6.7 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 512 megabytes (2^29 bytes), time= 12.8 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 1 gigabyte (2^30 bytes), time= 24.9 seconds
  no anomalies in 183 test result(s)
```

Universitat d'Alacant
Universidad de Alicante

```
rng=RNG_stdin, seed=0xe40f5ee7
length= 2 gigabytes (2^31 bytes), time= 48.1 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 4 gigabytes (2^32 bytes), time= 91.8 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]Gap-16:A                  R=  +4.8  p =  1.2e-3   unusual
  ...and 202 test result(s) without anomalies

rng=RNG_stdin, seed=0xe40f5ee7
length= 8 gigabytes (2^33 bytes), time= 182 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 16 gigabytes (2^34 bytes), time= 361 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 32 gigabytes (2^35 bytes), time= 710 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 64 gigabytes (2^36 bytes), time= 1434 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 128 gigabytes (2^37 bytes), time= 2883 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 256 gigabytes (2^38 bytes), time= 5650 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]BCFN(2+3,13-0,T)         R=  +8.4  p =  5.0e-4   unusual
  ...and 266 test result(s) without anomalies

rng=RNG_stdin, seed=0xe40f5ee7
length= 512 gigabytes (2^39 bytes), time= 11413 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0xe40f5ee7
length= 1 terabyte (2^40 bytes), time= 22887 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d11.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0xd6b4be6d
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0xd6b4be6d
length= 64 megabytes (2^26 bytes), time= 2.5 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 128 megabytes (2^27 bytes), time= 6.6 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 256 megabytes (2^28 bytes), time= 13.9 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/32]DC6-9x1Bytes-1           R=  +5.6  p =  7.9e-3   unusual
  ...and 161 test result(s) without anomalies

rng=RNG_stdin, seed=0xd6b4be6d
length= 512 megabytes (2^29 bytes), time= 26.0 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]BCFN(2+1,13-3,T)         R=  +8.7  p =  6.8e-4   unusual
  ...and 170 test result(s) without anomalies

rng=RNG_stdin, seed=0xd6b4be6d
length= 1 gigabyte (2^30 bytes), time= 48.3 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 2 gigabytes (2^31 bytes), time= 93.3 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 4 gigabytes (2^32 bytes), time= 182 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 8 gigabytes (2^33 bytes), time= 372 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 16 gigabytes (2^34 bytes), time= 744 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 32 gigabytes (2^35 bytes), time= 1437 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 64 gigabytes (2^36 bytes), time= 22822 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 128 gigabytes (2^37 bytes), time= 25757 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 256 gigabytes (2^38 bytes), time= 31401 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 512 gigabytes (2^39 bytes), time= 43127 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0xd6b4be6d
length= 1 terabyte (2^40 bytes), time= 66213 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d12.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x365ce5c9
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x365ce5c9
length= 64 megabytes (2^26 bytes), time= 2.5 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 128 megabytes (2^27 bytes), time= 6.4 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 256 megabytes (2^28 bytes), time= 13.0 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 512 megabytes (2^29 bytes), time= 23.5 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/32]DC6-9x1Bytes-1           R=  +5.6  p =  6.9e-3   unusual
  ...and 170 test result(s) without anomalies

rng=RNG_stdin, seed=0x365ce5c9
length= 1 gigabyte (2^30 bytes), time= 46.0 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 2 gigabytes (2^31 bytes), time= 89.8 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]FPF-14+6/16:all           R=  +4.8  p =  5.8e-4   unusual
  ...and 193 test result(s) without anomalies

rng=RNG_stdin, seed=0x365ce5c9
length= 4 gigabytes (2^32 bytes), time= 188 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 8 gigabytes (2^33 bytes), time= 432 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 16 gigabytes (2^34 bytes), time= 936 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]DC6-9x1Bytes-1            R=  +5.3  p =  5.9e-3   unusual
  ...and 225 test result(s) without anomalies

rng=RNG_stdin, seed=0x365ce5c9
length= 32 gigabytes (2^35 bytes), time= 1917 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 64 gigabytes (2^36 bytes), time= 23866 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 128 gigabytes (2^37 bytes), time= 27865 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0x365ce5c9
length= 256 gigabytes (2^38 bytes), time= 35526 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]DC6-9x1Bytes-1           R=  -4.3  p =1-5.2e-3   unusual
  ...and 266 test result(s) without anomalies

rng=RNG_stdin, seed=0x365ce5c9
length= 512 gigabytes (2^39 bytes), time= 51626 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]DC6-9x1Bytes-1           R=  -4.5  p =1-8.2e-3   unusual
  ...and 278 test result(s) without anomalies

rng=RNG_stdin, seed=0x365ce5c9
length= 1 terabyte (2^40 bytes), time= 75218 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d13.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x6517e441
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x6517e441
length= 64 megabytes (2^26 bytes), time= 2.6 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 128 megabytes (2^27 bytes), time= 6.0 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 256 megabytes (2^28 bytes), time= 12.0 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low4/32]FPF-14+6/16:all          R=  +4.5  p =  1.0e-3   unusual
  ...and 161 test result(s) without anomalies

rng=RNG_stdin, seed=0x6517e441
length= 512 megabytes (2^29 bytes), time= 22.1 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 1 gigabyte (2^30 bytes), time= 45.7 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 2 gigabytes (2^31 bytes), time= 91.0 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 4 gigabytes (2^32 bytes), time= 191 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 8 gigabytes (2^33 bytes), time= 395 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 16 gigabytes (2^34 bytes), time= 801 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 32 gigabytes (2^35 bytes), time= 1827 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 64 gigabytes (2^36 bytes), time= 25826 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 128 gigabytes (2^37 bytes), time= 33934 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 256 gigabytes (2^38 bytes), time= 49610 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 512 gigabytes (2^39 bytes), time= 73838 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0x6517e441
length= 1 terabyte (2^40 bytes), time= 96300 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d14.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x4b94f920
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x4b94f920
length= 64 megabytes (2^26 bytes), time= 2.4 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 128 megabytes (2^27 bytes), time= 6.0 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 256 megabytes (2^28 bytes), time= 11.9 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 512 megabytes (2^29 bytes), time= 22.3 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 1 gigabyte (2^30 bytes), time= 41.9 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 2 gigabytes (2^31 bytes), time= 79.5 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 4 gigabytes (2^32 bytes), time= 163 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 8 gigabytes (2^33 bytes), time= 400 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 16 gigabytes (2^34 bytes), time= 926 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]DC6-9x1Bytes-1            R=  -4.3  p =1-4.6e-3   unusual
  ...and 225 test result(s) without anomalies

rng=RNG_stdin, seed=0x4b94f920
length= 32 gigabytes (2^35 bytes), time= 1979 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 64 gigabytes (2^36 bytes), time= 24062 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 128 gigabytes (2^37 bytes), time= 28361 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]BCFN(2+0,13-0,T)          R=  +7.3  p =  1.9e-3   unusual
  ...and 257 test result(s) without anomalies

rng=RNG_stdin, seed=0x4b94f920
length= 256 gigabytes (2^38 bytes), time= 36553 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 512 gigabytes (2^39 bytes), time= 53722 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0x4b94f920
length= 1 terabyte (2^40 bytes), time= 77241 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d15.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x57aa39d1
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x57aa39d1
length= 64 megabytes (2^26 bytes), time= 3.0 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 128 megabytes (2^27 bytes), time= 7.1 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 256 megabytes (2^28 bytes), time= 14.0 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 512 megabytes (2^29 bytes), time= 26.0 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 1 gigabyte (2^30 bytes), time= 49.4 seconds
  Test Name                         Raw       Processed     Evaluation
  Gap-16:A                          R=  +5.1  p =  6.1e-4   unusual
  [Low4/32]BCFN(2+0,13-3,T)         R=  +7.7  p =  1.9e-3   unusual
  ...and 181 test result(s) without anomalies

rng=RNG_stdin, seed=0x57aa39d1
length= 2 gigabytes (2^31 bytes), time= 101 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 4 gigabytes (2^32 bytes), time= 202 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 8 gigabytes (2^33 bytes), time= 413 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 16 gigabytes (2^34 bytes), time= 822 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 32 gigabytes (2^35 bytes), time= 1898 seconds
  Test Name                         Raw       Processed     Evaluation
  [Low1/8]BCFN(2+1,13-0,T)          R=  -9.1  p =1-8.2e-5   unusual
  [Low1/32]BCFN(2+2,13-1,T)         R=  +8.3  p =  6.7e-4   unusual
  ...and 233 test result(s) without anomalies

rng=RNG_stdin, seed=0x57aa39d1
length= 64 gigabytes (2^36 bytes), time= 25432 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 128 gigabytes (2^37 bytes), time= 32590 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 256 gigabytes (2^38 bytes), time= 46527 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0x57aa39d1
length= 512 gigabytes (2^39 bytes), time= 71169 seconds
  Test Name                         Raw       Processed     Evaluation
  Gap-16:B                          R=  -4.1  p =1-2.0e-3   unusual
  ...and 278 test result(s) without anomalies

rng=RNG_stdin, seed=0x57aa39d1
length= 1 terabyte (2^40 bytes), time= 95160 seconds
  no anomalies in 290 test result(s)
```

ConSBox_d19.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x8aada72a
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x8aada72a
length= 64 megabytes (2^26 bytes), time= 3.5 seconds
  no anomalies in 139 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 128 megabytes (2^27 bytes), time= 8.1 seconds
  no anomalies in 151 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 256 megabytes (2^28 bytes), time= 16.5 seconds
  no anomalies in 162 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 512 megabytes (2^29 bytes), time= 30.9 seconds
  no anomalies in 171 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 1 gigabyte (2^30 bytes), time= 59.4 seconds
  no anomalies in 183 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 2 gigabytes (2^31 bytes), time= 114 seconds
  no anomalies in 194 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 4 gigabytes (2^32 bytes), time= 221 seconds
  no anomalies in 203 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 8 gigabytes (2^33 bytes), time= 446 seconds
  no anomalies in 215 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 16 gigabytes (2^34 bytes), time= 870 seconds
  no anomalies in 226 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 32 gigabytes (2^35 bytes), time= 2294 seconds
  no anomalies in 235 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 64 gigabytes (2^36 bytes), time= 26559 seconds
  no anomalies in 247 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 128 gigabytes (2^37 bytes), time= 35143 seconds
  no anomalies in 258 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 256 gigabytes (2^38 bytes), time= 51973 seconds
  no anomalies in 267 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 512 gigabytes (2^39 bytes), time= 75780 seconds
  no anomalies in 279 test result(s)

rng=RNG_stdin, seed=0x8aada72a
length= 1 terabyte (2^40 bytes), time= 98719 seconds
  no anomalies in 290 test result(s)
```

Test PractRand sin *s-box* $p = 65537, r = 1, s = 2$

SinSBox_t7.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0xfa47a924
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0xfa47a924
length= 64 megabytes (2^26 bytes), time= 3.6 seconds
  Test Name                         Raw       Processed     Evaluation
  FPF-14+6/16:cross                 R= +66.0  p =  2.0e-58    FAIL !!!!
  ...and 138 test result(s) without anomalies
```

SinSBox_t15.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x6e349e46
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x6e349e46
length= 64 megabytes (2^26 bytes), time= 3.9 seconds
  Test Name                         Raw       Processed     Evaluation
  FPF-14+6/16:cross                 R= +64.4  p =  5.3e-57    FAIL !!!!
  ...and 138 test result(s) without anomalies
```

SinSBox_t13.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x1cf730af
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x1cf730af
length= 32 megabytes (2^25 bytes), time= 2.1 seconds
  Test Name                         Raw       Processed     Evaluation
  FPF-14+6/16:cross                 R= +22.0  p =  8.0e-18    FAIL !
  ...and 129 test result(s) without anomalies
```

SinSBox_t19.txt

```
RNG = RNG_stdin, PractRand version 0.92, seed = 0x885ce025
test set = normal, folding = standard(unknown format)

rng=RNG_stdin, seed=0x885ce025
length= 32 megabytes (2^25 bytes), time= 2.1 seconds
  Test Name                         Raw       Processed     Evaluation
  FPF-14+6/16:cross                 R= +32.2  p =  8.5e-26    FAIL !!
  ...and 129 test result(s) without anomalies
```

Universitat d'Alacant
Universidad de Alicante

TestU01

Test pasados satisfactoriamente para los cinco conjuntos de test, y diez claves distintas, en los generadores M7, M10 y M19.

Identificadores (0-9) para 10 claves distintas

M7

	alphabit	rabbit	scrush	mcrush	bcrush
0	17	33	15	144	160
1	17	33	15	143(4)	160
2	17	33	15	143(4)	160
3	17	33	15	144	160
4	17	33	15	144	160
5	17	33	15	144	160
6	17	32(1)	15	144	160
7	17	32(1)	15	144	160
8	17	32(1)	15	142(7)(3)	160
9	17	33	15	143(8)	160

M10

	alphabit	rabbit	scrush	mcrush	bcrush
0	17	33	15	144	160
1	17	33	15	144	160
2	17	33	15	144	160
3	17	32(1)	15	144	159(2)
4	17	33	15	144	160
5	17	33	15	144	160
6	17	33	15	143(3)	160
7	17	32(1)	15	144	160
8	17	33	15	144	160
9	17	33	15	142(9)(6)	160

M19

	alphabit	rabbit	scrush	mcrush	bcrush
0	17	33	15	144	160
1	16(3)	32(3)	15	144	160
2	17	33	15	144	160
3	17	33	15	142(0)(4)	158(4)(5)
4	17	33	15	144	160
5	17	32(1)	15	144	160
6	17	33	15	144	160
7	17	33	15	144	160
8	16(3)	33	15	144	159(6)
9	17	33	15	144	160

Leyenda de test con algún valor fuera de rango:

(0) SerialOver (1) MultinomialBitsOver (2) Run (3) RandomWalk1 (4) CollisionOver (5) GCD (6) Gap (7) MatrixRank (8) MaxOft (9) CouponCollector

Ejemplo de resultado de análisis de TestU01 para M10 con el conjunto de test Big Crush:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Starting BigCrush
                Version:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


************************************************************
Test smarsa_SerialOver calling smultin_MultinomialOver
************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
-----------------------------------------------
   N =  1,  n = 1000000000,  r =  0,   d =  256,   t =  3,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =          16777216
       Expected number per cell =   59.604645
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =   0.0083558402,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          : 16711680
Value of the statistic                : 1.67e+7
p-value of test                       :    0.14


-----------------------------------------------
CPU time used                    :  00:02:32.65

Generator state:




************************************************************
Test smarsa_SerialOver calling smultin_MultinomialOver
************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
-----------------------------------------------
   N =  1,  n = 1000000000,  r = 22,   d =  256,   t =  3,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =          16777216
       Expected number per cell =   59.604645
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =   0.0083558402,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          : 16711680
Value of the statistic                : 1.67e+7
p-value of test                       :    0.95


-----------------------------------------------
CPU time used                    :  00:02:54.51

Generator state:
```

```
************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver
************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
-----------------------------------------------
   N = 30,  n = 20000000,  r =  0,   d = 2097152,   t =  2,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47

-----------------------------------------------
Results of CollisionOver test:

POISSON approximation          :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1361
p-value of test                       :     0.53

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334481
  j =  1                              :       599997278
  j =  2                              :            1361
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                    :  00:03:53.56

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver
************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
-----------------------------------------------
   N = 30,  n = 20000000,  r =  9,   d = 2097152,   t =  2,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47
```

```
-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1351
p-value of test                       :    0.63

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334471
  j =  1                              :       599997298
  j =  2                              :            1351
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                 :  00:03:58.64

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r =  0,   d = 16384,   t =  3,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1306
p-value of test                       :    0.94

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334426
  j =  1                              :       599997388
  j =  2                              :            1306
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                 :  00:04:47.45

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r = 16,   d = 16384,   t =  3,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1356
p-value of test                       :    0.58

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334476
  j =  1                              :       599997288
  j =  2                              :            1356
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                 :  00:04:52.12

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r =  0,   d =   64,   t =  7,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1431
p-value of test                       :    0.04

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334551
  j =  1                              :       599997138
  j =  2                              :            1431
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                 :  00:04:50.25

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r = 24,   d =   64,   t =  7,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1407
p-value of test                       :    0.13

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334527
  j =  1                              :       599997186
  j =  2                              :            1407
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

-----------------------------------------------
CPU time used                 :  00:04:54.78

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r =  0,   d =    8,   t = 14,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


-----------------------------------------------
Results of CollisionOver test:

POISSON approximation                 :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1406
p-value of test                       :    0.13
```

```
------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334526
  j =  1                              :       599997188
  j =  2                              :            1406
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

------------------------------------------------
CPU time used                     :  00:04:49.78

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r = 27,   d =    8,   t = 14,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


------------------------------------------------
Results of CollisionOver test:

POISSON approximation             :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1386
p-value of test                       :    0.28

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334506
  j =  1                              :       599997228
  j =  2                              :            1386
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

------------------------------------------------
CPU time used                     :  00:04:54.68

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r =  0,   d =    4,   t = 21,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


------------------------------------------------
Results of CollisionOver test:

POISSON approximation             :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1359
p-value of test                       :    0.55

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334479
  j =  1                              :       599997282
  j =  2                              :            1359
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

------------------------------------------------
CPU time used                     :  00:04:49.53

Generator state:



************************************************************
Test smarsa_CollisionOver calling smultin_MultinomialOver

************************************************************
HOST =

GenP2


smultin_MultinomialOver test:
----------------------------------------------
   N = 30,  n = 20000000,  r = 28,   d =    4,   t = 21,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellSerial
       Number of cells = d^t =      4398046511104
       Expected number per cell =  1 /  219902.33
       EColl = n^2 / (2k) =  45.47473509
       Hashing =   TRUE

       Collision test

       CollisionOver:   density = n / k =  1 /  219902.33
       Expected number of collisions = Mu =      45.47


------------------------------------------------
Results of CollisionOver test:

POISSON approximation             :
Expected number of collisions = N*Mu  :     1364.24
Observed number of collisions         :     1399
p-value of test                       :    0.18

------------------------------
Total number of cells containing j balls

  j =  0                              :  131940795334519
  j =  1                              :       599997202
  j =  2                              :            1399
  j =  3                              :               0
  j =  4                              :               0
  j =  5                              :               0

------------------------------------------------
CPU time used                     :  00:04:53.75

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
----------------------------------------------
   N = 250,  n = 4000000,  r =  0,    d = 1073741824,    t = 2,
p = 1

      Number of cells = d^t = 1152921504606846976
      Lambda = Poisson mean =      13.8778


-----------------------------------------------------
Total expected number = N*Lambda      :    3469.45
Total observed number                 :    3507
p-value of test                       :    0.26


------------------------------------------------
CPU time used                     :  00:05:31.75

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
----------------------------------------------
   N = 20,  n = 20000000,  r =  0,    d = 2097152,    t = 3,    p =
1

      Number of cells = d^t = 9223372036854775808
      Lambda = Poisson mean =     216.8404

-----------------------------------------------------
Total expected number = N*Lambda      :    4336.81
Total observed number                 :    4265
p-value of test                       :    0.86


------------------------------------------------
CPU time used                     :  00:02:42.76

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
----------------------------------------------
   N = 20,  n = 30000000,  r = 14,    d = 65536,    t = 4,    p = 1

      Number of cells = d^t = 18446744073709551616
      Lambda = Poisson mean =     365.9182


-----------------------------------------------------
Total expected number = N*Lambda      :    7318.36
Total observed number                 :    7257
p-value of test                       :    0.76


------------------------------------------------
CPU time used                     :  00:04:37.48

Generator state:



************************************************************
HOST =

GenP2
```

```
smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 20000000,  r =  0,    d = 512,    t = 7,    p = 1


      Number of cells = d^t = 9223372036854775808
      Lambda = Poisson mean =     216.8404


---------------------------------------------------
Total expected number = N*Lambda      :    4336.81
Total observed number                 :    4384
p-value of test                       :    0.24


-----------------------------------------------
CPU time used                    :  00:03:56.75

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 20000000,  r =  7,    d = 512,    t = 7,    p = 1


      Number of cells = d^t = 9223372036854775808
      Lambda = Poisson mean =     216.8404


---------------------------------------------------
Total expected number = N*Lambda      :    4336.81
Total observed number                 :    4442
p-value of test                       :    0.06


-----------------------------------------------
CPU time used                    :  00:04:04.21

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 30000000,  r = 14,    d = 256,    t = 8,    p = 1


      Number of cells = d^t = 18446744073709551616
      Lambda = Poisson mean =     365.9182


---------------------------------------------------
Total expected number = N*Lambda      :    7318.36
Total observed number                 :    7362
p-value of test                       :    0.31


-----------------------------------------------
CPU time used                    :  00:06:37.87

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 30000000,  r = 22,    d = 256,    t = 8,    p = 1


      Number of cells = d^t = 18446744073709551616
      Lambda = Poisson mean =     365.9182


---------------------------------------------------
Total expected number = N*Lambda      :    7318.36
Total observed number                 :    7321
p-value of test                       :    0.49


-----------------------------------------------
CPU time used                    :  00:06:37.98

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 30000000,  r =  0,    d = 16,    t = 16,    p = 1


      Number of cells = d^t = 18446744073709551616
      Lambda = Poisson mean =     365.9182


---------------------------------------------------
Total expected number = N*Lambda      :    7318.36
Total observed number                 :    7303
p-value of test                       :    0.57


-----------------------------------------------
CPU time used                    :  00:10:20.79

Generator state:



************************************************************
HOST =

GenP2


smarsa_BirthdaySpacings test:
-----------------------------------------------
   N = 20,  n = 30000000,  r = 26,    d = 16,    t = 16,    p = 1


      Number of cells = d^t = 18446744073709551616
      Lambda = Poisson mean =     365.9182


---------------------------------------------------
Total expected number = N*Lambda      :    7318.36
Total observed number                 :    7379
p-value of test                       :    0.24


-----------------------------------------------
CPU time used                    :  00:10:40.76

Generator state:



************************************************************
HOST =

GenP2


snpair_ClosePairs test:
-----------------------------------------------
   N = 30,  n = 6000000,  r =  0,  t = 3,  p = 0,  m = 30,  Torus =
TRUE


---------------------------------------
Test based on the 2 nearest points (NP):

Stat. AD on the N values (NP)         :    1.62
p-value of test                       :    0.15


A2 test based on the spacings between the
   successive jump times of process Y_n(t):

A2 test on the values of A2 (m-NP)    :    0.67
p-value of test                       :    0.58

Test on the Nm values of W_{n,i}(mNP1):    2.79
p-value of test                       :    0.03

Test on the jump times of Y
   (superposition of Yn):

Expected number of jumps of Y = mN    :     900
Number of jumps of Y                  :     955
p-value of test                       :    0.04

Stat. AD (mNP2)                       :    2.00
p-value of test                       :    0.09

Stat. AD after spacings (mNP2-S)      :    1.36
p-value of test                       :    0.21

-----------------------------------------------
CPU time used                    :  00:03:30.51

Generator state:



************************************************************
HOST =

GenP2


snpair_ClosePairs test:
-----------------------------------------------
   N = 20,  n = 4000000,  r =  0,  t = 5,  p = 0,  m = 30,  Torus =
TRUE


---------------------------------------
Test based on the 2 nearest points (NP):

Stat. AD on the N values (NP)         :    2.53
p-value of test                       :    0.05


A2 test based on the spacings between the
   successive jump times of process Y_n(t):

A2 test on the values of A2 (m-NP)    :    0.89
p-value of test                       :    0.42

Test on the Nm values of W_{n,i}(mNP1):    2.15
p-value of test                       :    0.08

Test on the jump times of Y
   (superposition of Yn):

Expected number of jumps of Y = mN    :     600
Number of jumps of Y                  :     671
p-value of test                       :  2.3e-3

Stat. AD (mNP2)                       :    1.06
p-value of test                       :    0.33

Stat. AD after spacings (mNP2-S)      :    1.67
p-value of test                       :    0.14

-----------------------------------------------
CPU time used                    :  00:02:21.18

Generator state:



************************************************************
```

Universitat d'Alacant
Universidad de Alicante

```
HOST =

GenP2


snpair_ClosePairs test:
-----------------------------------------------
   N = 10,  n = 3000000,  r =  0,  t = 9,  p = 0,  m = 30,  Torus =
TRUE


---------------------------------------
Test based on the 2 nearest points (NP):

Stat. AD on the N values (NP)         :    0.48
p-value of test                       :    0.77


A2 test based on the spacings between the
   successive jump times of process Y_n(t):

A2 test on the values of A2 (m-NP)    :    1.66
p-value of test                       :    0.14

Test on the Nm values of W_{n,i}(mNP1):    0.85
p-value of test                       :    0.45

Test on the jump times of Y
   (superposition of Yn):

Expected number of jumps of Y = mN    :     300
Number of jumps of Y                  :     303
p-value of test                       :    0.44

Stat. AD (mNP2)                       :    0.91
p-value of test                       :    0.41

Stat. AD after spacings (mNP2-S)      :    0.62
p-value of test                       :    0.63

-----------------------------------------------
CPU time used                    :  00:03:34.60

Generator state:



************************************************************
HOST =

GenP2


snpair_ClosePairs test:
-----------------------------------------------
   N =  5,  n = 2000000,  r =  0,  t = 16,  p = 0,  m = 30,  Torus
=  TRUE


---------------------------------------
Test based on the 2 nearest points (NP):

Stat. AD on the N values (NP)         :    1.26
p-value of test                       :    0.24


A2 test based on the spacings between the
   successive jump times of process Y_n(t):

A2 test on the values of A2 (m-NP)    :    1.75
p-value of test                       :    0.13

Test on the Nm values of W_{n,i}(mNP1):    0.67
p-value of test                       :    0.58

Test on the jump times of Y
   (superposition of Yn):

Expected number of jumps of Y = mN    :     150
Number of jumps of Y                  :     154
p-value of test                       :    0.38

Stat. AD (mNP2)                       :    0.52
p-value of test                       :    0.73

Stat. AD after spacings (mNP2-S)      :    1.19
p-value of test                       :    0.27

-----------------------------------------------
CPU time used                    :  00:03:55.65

Generator state:



************************************************************
HOST =

GenP2


sknuth_SimpPoker test:
-----------------------------------------------
   N =  1,  n = 400000000,  r =  0,   d =    8,   k =    8


-----------------------------------------------
Number of degrees of freedom          :    7
Chi-square statistic                  :    8.19
p-value of test                       :    0.32

-----------------------------------------------
CPU time used                    :  00:02:57.57

Generator state:



************************************************************
HOST =

GenP2


sknuth_SimpPoker test:
-----------------------------------------------
   N =  1,  n = 400000000,  r = 27,   d =    8,   k =    8


-----------------------------------------------
Number of degrees of freedom          :    7
Chi-square statistic                  :    4.49
p-value of test                       :    0.72

-----------------------------------------------
CPU time used                    :  00:03:14.32

Generator state:



************************************************************
HOST =

GenP2


sknuth_SimpPoker test:
-----------------------------------------------
   N =  1,  n = 100000000,  r =  0,   d =   32,   k =   32


-----------------------------------------------
Number of degrees of freedom          :   18
Chi-square statistic                  :   24.98
p-value of test                       :    0.13

-----------------------------------------------
CPU time used                    :  00:02:53.29

Generator state:



************************************************************
HOST =

GenP2


sknuth_SimpPoker test:
-----------------------------------------------
   N =  1,  n = 100000000,  r = 25,   d =   32,   k =   32


-----------------------------------------------
Number of degrees of freedom          :   18
Chi-square statistic                  :    8.80
p-value of test                       :    0.96

-----------------------------------------------
CPU time used                    :  00:03:11.00

Generator state:



************************************************************
HOST =

GenP2


sknuth_CouponCollector test:
-----------------------------------------------
   N =  1,  n = 200000000,  r =  0,   d =    8


-----------------------------------------------
Number of degrees of freedom          :   54
Chi-square statistic                  :   41.34
p-value of test                       :    0.90

-----------------------------------------------
CPU time used                    :  00:03:53.67

Generator state:



************************************************************
HOST =

GenP2


sknuth_CouponCollector test:
-----------------------------------------------
   N =  1,  n = 200000000,  r = 10,   d =    8


-----------------------------------------------
Number of degrees of freedom          :   54
Chi-square statistic                  :   58.89
p-value of test                       :    0.30

-----------------------------------------------
CPU time used                    :  00:04:14.00

Generator state:



************************************************************
HOST =

GenP2


sknuth_CouponCollector test:
-----------------------------------------------
   N =  1,  n = 200000000,  r = 20,   d =    8


-----------------------------------------------
Number of degrees of freedom          :   54
Chi-square statistic                  :   41.70
p-value of test                       :    0.89

-----------------------------------------------
CPU time used                    :  00:04:13.73

Generator state:



************************************************************
HOST =
```

```
GenP2

sknuth_CouponCollector test:
-----------------------------------------------
   N =  1,  n = 200000000,  r = 27,   d =    8

-----------------------------------------------
Number of degrees of freedom          :   54
Chi-square statistic                  :   58.55
p-value of test                       :    0.31

-----------------------------------------------
CPU time used                    :  00:04:13.89

Generator state:



************************************************************
HOST =

GenP2

sknuth_Gap test:
-----------------------------------------------
   N =  1,  n = 500000000,  r =  0,   Alpha =       0,   Beta  =
0.0625

-----------------------------------------------
Number of degrees of freedom          :  232
Chi-square statistic                  :  225.79
p-value of test                       :    0.60

-----------------------------------------------
CPU time used                    :  00:06:20.07

Generator state:



************************************************************
HOST =

GenP2

sknuth_Gap test:
-----------------------------------------------
   N =  1,  n = 300000000,  r = 25,   Alpha =       0,   Beta  =
0.03125

-----------------------------------------------
Number of degrees of freedom          :  434
Chi-square statistic                  :  413.24
p-value of test                       :    0.76

-----------------------------------------------
CPU time used                    :  00:07:40.17

Generator state:



************************************************************
HOST =

GenP2

sknuth_Gap test:
-----------------------------------------------
   N =  1,  n = 100000000,  r =  0,   Alpha =       0,   Beta  =
0.0078125

-----------------------------------------------
Number of degrees of freedom          : 1437
Chi-square statistic                  : 1461.29
p-value of test                       :    0.32

-----------------------------------------------
CPU time used                    :  00:09:57.14

Generator state:



************************************************************
HOST =

GenP2

sknuth_Gap test:
-----------------------------------------------
   N =  1,  n = 10000000,  r = 20,   Alpha =       0,   Beta  =
0.000976563

-----------------------------------------------
Number of degrees of freedom          : 7046
Chi-square statistic                  : 6970.28
p-value of test                       :    0.74

-----------------------------------------------
CPU time used                    :  00:08:04.73

Generator state:



************************************************************
HOST =

GenP2

sknuth_Run test:
-----------------------------------------------
   N =  5,  n = 1000000000,  r =  0,   Up = FALSE

-----------------------------------------------
Kolmogorov-Smirnov+ statistic = D+    :   0.073
p-value of test                       :    0.90

Kolmogorov-Smirnov- statistic = D-    :    0.35
p-value of test                       :    0.24

Anderson-Darling statistic = A2       :    0.83
p-value of test                       :    0.45

Test on the sum of all N observations
Number of degrees of freedom          :   30
Chi-square statistic                  :   37.34
p-value of test                       :    0.17

-----------------------------------------------
CPU time used                    :  00:04:28.84

Generator state:



************************************************************
HOST =

GenP2

sknuth_Run test:
-----------------------------------------------
   N = 10,  n = 1000000000,  r = 15,   Up =  TRUE

-----------------------------------------------
Kolmogorov-Smirnov+ statistic = D+    :   0.093
p-value of test                       :    0.79

Kolmogorov-Smirnov- statistic = D-    :    0.42
p-value of test                       :    0.02

Anderson-Darling statistic = A2       :    2.76
p-value of test                       :    0.04

Test on the sum of all N observations
Number of degrees of freedom          :   60
Chi-square statistic                  :   85.10
p-value of test                       :    0.02

-----------------------------------------------
CPU time used                    :  00:09:43.00

Generator state:



************************************************************
Test sknuth_Permutation calling smultin_Multinomial
************************************************************
HOST =

GenP2

smultin_Multinomial test:
-----------------------------------------------
   N =  1,  n = 1000000000,  r =  5,   t =  3,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =                6
       Expected number per cell =  1.6666667e+08
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =  2.5000002e-09,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          :    5
Value of the statistic                :    8.69
p-value of test                       :    0.12

-----------------------------------------------
CPU time used                    :  00:02:57.84

Generator state:



************************************************************
Test sknuth_Permutation calling smultin_Multinomial
************************************************************
HOST =

GenP2

smultin_Multinomial test:
-----------------------------------------------
   N =  1,  n = 1000000000,  r =  5,   t =  5,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =              120
       Expected number per cell =   8333333.3
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =  5.9500005e-08,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          :  119
Value of the statistic                :   94.98
p-value of test                       :    0.95


-----------------------------------------------
CPU time used                    :  00:05:04.51

Generator state:
```

```
************************************************************
Test sknuth_Permutation calling smultin_Multinomial

************************************************************
HOST =

GenP2


smultin_Multinomial test:
-----------------------------------------------
   N =  1,  n = 500000000,  r =  5,   t =  7,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =              5040
       Expected number per cell =   99206.349
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =  5.0390004e-06,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          : 5039
Value of the statistic                : 4963.79
p-value of test                       :    0.77


-----------------------------------------------
CPU time used                    :  00:03:42.95

Generator state:



************************************************************
Test sknuth_Permutation calling smultin_Multinomial

************************************************************
HOST =

GenP2


smultin_Multinomial test:
-----------------------------------------------
   N =  1,  n = 500000000,  r = 10,   t = 10,
       Sparse =  FALSE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =           3628800
       Expected number per cell =    137.7866
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =   0.0036287993,    Sigma =          1

-----------------------------------------------
Test Results for Delta =   1.0000

Number of degrees of freedom          : 3628799
Value of the statistic                : 3.63e+6
p-value of test                       :    0.18


-----------------------------------------------
CPU time used                    :  00:06:30.51

Generator state:



************************************************************
Test sknuth_CollisionPermut calling smultin_Multinomial

************************************************************
HOST =

GenP2


smultin_Multinomial test:
-----------------------------------------------
   N = 20,  n = 20000000,  r =  0,   t = 14,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =       87178291200
       Expected number per cell =  1 /  4358.9146
       EColl = n^2 / (2k) =   2294.14912
       Hashing =   TRUE

       Collision test,    Mu =      2293.9736,    Sigma =
47.8841

-----------------------------------------------
Test Results for Collisions

For the total number of collisions, we use
      the Poisson approximation:
Expected number of collisions = N*Mu  :    45879.47
Observed number of collisions         :    45895
p-value of test                       :    0.47


------------------------------
Total number of cells containing j balls

  j =  0                              :    1743165869895
  j =  1                              :        399908216
  j =  2                              :            45883
  j =  3                              :                6
  j =  4                              :                0
  j =  5                              :                0

-----------------------------------------------
CPU time used                    :  00:09:01.12

Generator state:



************************************************************
Test sknuth_CollisionPermut calling smultin_Multinomial
************************************************************
HOST =

GenP2


smultin_Multinomial test:
-----------------------------------------------
   N = 20,  n = 20000000,  r = 10,   t = 14,
       Sparse =   TRUE

       GenerCell = smultin_GenerCellPermut
       Number of cells = t! =       87178291200
       Expected number per cell =  1 /  4358.9146
       EColl = n^2 / (2k) =   2294.14912
       Hashing =   TRUE

       Collision test,    Mu =      2293.9736,    Sigma =
47.8841

-----------------------------------------------
Test Results for Collisions

For the total number of collisions, we use
      the Poisson approximation:
Expected number of collisions = N*Mu  :    45879.47
Observed number of collisions         :    45631
p-value of test                       :    0.88


------------------------------
Total number of cells containing j balls

  j =  0                              :    1743165869631
  j =  1                              :        399908743
  j =  2                              :            45621
  j =  3                              :                5
  j =  4                              :                0
  j =  5                              :                0

-----------------------------------------------
CPU time used                    :  00:09:06.62

Generator state:



************************************************************
HOST =

GenP2


sknuth_MaxOft test:
-----------------------------------------------
   N = 40,  n = 10000000,  r =  0,   d = 100000,   t =  8

       Number of categories = 100000
       Expected number per category  = 100.00


-----------------------------------------------
Test results for chi2 with 99999 degrees of freedom:

Kolmogorov-Smirnov+ statistic = D+    :    0.18
p-value of test                       :    0.07

Kolmogorov-Smirnov- statistic = D-    :   0.023
p-value of test                       :    0.94

Anderson-Darling statistic = A2       :    1.89
p-value of test                       :    0.11

Test on the sum of all N observations
Number of degrees of freedom          : 3999960
Chi-square statistic                  : 3.99e+6
p-value of test                       :    0.97

-----------------------------------------------
Test results for Anderson-Darling:

Kolmogorov-Smirnov+ statistic = D+    :   0.081
p-value of test                       :    0.56

Kolmogorov-Smirnov- statistic = D-    :   0.044
p-value of test                       :    0.83

Anderson-Darling statistic = A2       :    0.38
p-value of test                       :    0.87


-----------------------------------------------
CPU time used                    :  00:05:01.78

Generator state:



************************************************************
HOST =

GenP2


sknuth_MaxOft test:
-----------------------------------------------
   N = 30,  n = 10000000,  r =  0,   d = 100000,   t = 16

       Number of categories = 100000
       Expected number per category  = 100.00


-----------------------------------------------
Test results for chi2 with 99999 degrees of freedom:

Kolmogorov-Smirnov+ statistic = D+    :   0.068
p-value of test                       :    0.73

Kolmogorov-Smirnov- statistic = D-    :   0.099
p-value of test                       :    0.52

Anderson-Darling statistic = A2       :    0.33
p-value of test                       :    0.91

Test on the sum of all N observations
Number of degrees of freedom          : 2999970
Chi-square statistic                  : 3.00e+6
p-value of test                       :    0.39


-----------------------------------------------
Test results for Anderson-Darling:
```

Universitat d'Alacant
Universidad de Alicante

```
Kolmogorov-Smirnov+ statistic = D+    :   0.044
p-value of test                       :    0.87

Kolmogorov-Smirnov- statistic = D-    :    0.18
p-value of test                       :    0.12

Anderson-Darling statistic = A2       :    1.40
p-value of test                       :    0.20


------------------------------------------------
CPU time used                    :  00:05:48.51

Generator state:



************************************************************
HOST =

GenP2


sknuth_MaxOft test:
------------------------------------------------
   N = 20,  n = 10000000,  r =  0,   d = 100000,   t = 24

      Number of categories = 100000
      Expected number per category  = 100.00


------------------------------------------------
Test results for chi2 with 99999 degrees of freedom:

Kolmogorov-Smirnov+ statistic = D+    :    0.23
p-value of test                       :    0.10

Kolmogorov-Smirnov- statistic = D-    :   0.036
p-value of test                       :    0.93

Anderson-Darling statistic = A2       :    0.84
p-value of test                       :    0.45

Test on the sum of all N observations
Number of degrees of freedom          : 1999980
Chi-square statistic                  : 2.00e+6
p-value of test                       :    0.83


------------------------------------------------
Test results for Anderson-Darling:

Kolmogorov-Smirnov+ statistic = D+    : 7.33e-3
p-value of test                       :    0.9916

Kolmogorov-Smirnov- statistic = D-    :    0.20
p-value of test                       :    0.19

Anderson-Darling statistic = A2       :    1.07
p-value of test                       :    0.32


------------------------------------------------
CPU time used                    :  00:05:11.48

Generator state:



************************************************************
HOST =

GenP2


sknuth_MaxOft test:
------------------------------------------------
   N = 20,  n = 10000000,  r =  0,   d = 100000,   t = 32

      Number of categories = 100000
      Expected number per category  = 100.00


------------------------------------------------
Test results for chi2 with 99999 degrees of freedom:

Kolmogorov-Smirnov+ statistic = D+    :    0.17
p-value of test                       :    0.28

Kolmogorov-Smirnov- statistic = D-    :    0.16
p-value of test                       :    0.34

Anderson-Darling statistic = A2       :    0.87
p-value of test                       :    0.43

Test on the sum of all N observations
Number of degrees of freedom          : 1999980
Chi-square statistic                  : 2.00e+6
p-value of test                       :    0.68


------------------------------------------------
Test results for Anderson-Darling:

Kolmogorov-Smirnov+ statistic = D+    :   0.081
p-value of test                       :    0.73

Kolmogorov-Smirnov- statistic = D-    :    0.19
p-value of test                       :    0.21

Anderson-Darling statistic = A2       :    0.82
p-value of test                       :    0.46


------------------------------------------------
CPU time used                    :  00:06:32.92

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleProd test:
------------------------------------------------
   N = 40,  n = 10000000,  r =  0,   t = 8

------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.11
p-value of test                       :    0.33

Kolmogorov-Smirnov- statistic = D-    :   0.042
p-value of test                       :    0.84

Anderson-Darling statistic = A2       :    0.61
p-value of test                       :    0.64

------------------------------------------------
CPU time used                    :  00:03:58.96

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleProd test:
------------------------------------------------
   N = 20,  n = 10000000,  r =  0,   t = 16

------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.20
p-value of test                       :    0.16

Kolmogorov-Smirnov- statistic = D-    :    0.10
p-value of test                       :    0.62

Anderson-Darling statistic = A2       :    0.84
p-value of test                       :    0.45

------------------------------------------------
CPU time used                    :  00:03:26.01

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleProd test:
------------------------------------------------
   N = 20,  n = 10000000,  r =  0,   t = 24

------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.32
p-value of test                       :    0.01

Kolmogorov-Smirnov- statistic = D-    :   0.021
p-value of test                       :    0.97

Anderson-Darling statistic = A2       :    2.76
p-value of test                       :    0.04

------------------------------------------------
CPU time used                    :  00:04:52.14

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleMean test:
------------------------------------------------
   N = 20000000,  n = 30,  r =  0

------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    : 1.16e-4
p-value of test                       :    0.58

Kolmogorov-Smirnov- statistic = D-    : 1.25e-4
p-value of test                       :    0.54

Anderson-Darling statistic = A2       :    0.48
p-value of test                       :    0.77

------------------------------------------------
CPU time used                    :  00:00:47.85

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleMean test:
------------------------------------------------
   N = 20000000,  n = 30,  r = 10

------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    : 2.04e-4
p-value of test                       :    0.19

Kolmogorov-Smirnov- statistic = D-    : 1.19e-4
p-value of test                       :    0.57

Anderson-Darling statistic = A2       :    0.52
p-value of test                       :    0.73

------------------------------------------------
```

Universitat d'Alacant
Universidad de Alicante

```
CPU time used                    :  00:00:48.39

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleCorr test:
-----------------------------------------------
   N =  1,  n = 2000000000,  r =  0,   k = 1


-----------------------------------------------
Normal statistic                 :    0.54
p-value of test                  :    0.29

-----------------------------------------------
CPU time used                    :  00:01:35.50

Generator state:



************************************************************
HOST =

GenP2


svaria_SampleCorr test:
-----------------------------------------------
   N =  1,  n = 2000000000,  r =  0,   k = 2


-----------------------------------------------
Normal statistic                 :    0.96
p-value of test                  :    0.17

-----------------------------------------------
CPU time used                    :  00:01:35.53

Generator state:



************************************************************
HOST =

GenP2


svaria_AppearanceSpacings test:
-----------------------------------------------
   N =  1,   Q = 10000000,   K = 1000000000,   r = 0,   s = 3,   L
= 15

   Sequences of n = (K + Q)L =  15150000000 bits
   Q = 10000000 initialization blocks
   K = 1000000000 blocks for the test
   the blocks have L = 15 bits


-----------------------------------------------
Normal statistic                 :    0.83
p-value of test                  :    0.20

-----------------------------------------------
CPU time used                    :  00:04:13.45

Generator state:



************************************************************
HOST =

GenP2


svaria_AppearanceSpacings test:
-----------------------------------------------
   N =  1,   Q = 10000000,   K = 1000000000,   r = 27,   s = 3,   L
= 15

   Sequences of n = (K + Q)L =  15150000000 bits
   Q = 10000000 initialization blocks
   K = 1000000000 blocks for the test
   the blocks have L = 15 bits


-----------------------------------------------
Normal statistic                 :   -0.23
p-value of test                  :    0.59

-----------------------------------------------
CPU time used                    :  00:04:11.62

Generator state:



************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r =  0,  k = 256,  Alpha =      0,  Beta
=   0.25


-----------------------------------------------
Number of degrees of freedom     :   67
Chi-square statistic             :   72.98
p-value of test                  :    0.29

-----------------------------------------------
CPU time used                    :  00:04:34.15

Generator state:



************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r = 20,  k = 256,  Alpha =      0,  Beta
=   0.25


-----------------------------------------------
Number of degrees of freedom     :   67
Chi-square statistic             :   62.57
p-value of test                  :    0.63

-----------------------------------------------
CPU time used                    :  00:04:43.48

Generator state:



************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r = 28,  k = 256,  Alpha =      0,  Beta
=   0.25


-----------------------------------------------
Number of degrees of freedom     :   67
Chi-square statistic             :   48.46
p-value of test                  :    0.96

-----------------------------------------------
CPU time used                    :  00:04:43.50

Generator state:



************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r =  0,  k = 256,  Alpha =      0,  Beta
= 0.0625


-----------------------------------------------
Number of degrees of freedom     :   37
Chi-square statistic             :   44.53
p-value of test                  :    0.18

-----------------------------------------------
CPU time used                    :  00:04:19.03

Generator state:

************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r = 10,  k = 256,  Alpha =      0,  Beta
= 0.0625


-----------------------------------------------
Number of degrees of freedom     :   37
Chi-square statistic             :   34.01
p-value of test                  :    0.61

-----------------------------------------------
CPU time used                    :  00:04:24.34

Generator state:



************************************************************
HOST =

GenP2


svaria_WeightDistrib test:
-----------------------------------------------
   N =  1,  n = 20000000,  r = 26,  k = 256,  Alpha =      0,  Beta
= 0.0625


-----------------------------------------------
Number of degrees of freedom     :   37
Chi-square statistic             :   38.07
p-value of test                  :    0.42

-----------------------------------------------
CPU time used                    :  00:04:24.54

Generator state:



************************************************************
HOST =

GenP2
```

Universitat d'Alacant
Universidad de Alicante

```
svaria_SumCollector test:
------------------------------------------------
   N =  1,  n = 500000000,  r =  0,   g = 10


------------------------------------------------
Number of degrees of freedom          :   29
Chi-square statistic                  :   21.72
p-value of test                       :    0.83

------------------------------------------------
CPU time used                    :  00:08:52.31

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N = 10,  n = 1000000,  r =  0,    s = 5,    L = 30,    k = 30


------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :   0.059
p-value of test                       :    0.90

Kolmogorov-Smirnov- statistic = D-    :    0.20
p-value of test                       :    0.40

Anderson-Darling statistic = A2       :    0.42
p-value of test                       :    0.83

Test on the sum of all N observations
Number of degrees of freedom          :   40
Chi-square statistic                  :   45.18
p-value of test                       :    0.26

------------------------------------------------
CPU time used                    :  00:02:23.32

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N = 10,  n = 1000000,  r = 25,    s = 5,    L = 30,    k = 30


------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.12
p-value of test                       :    0.71

Kolmogorov-Smirnov- statistic = D-    :    0.23
p-value of test                       :    0.31

Anderson-Darling statistic = A2       :    0.45
p-value of test                       :    0.80

Test on the sum of all N observations
Number of degrees of freedom          :   40
Chi-square statistic                  :   42.28
p-value of test                       :    0.37

------------------------------------------------
CPU time used                    :  00:02:23.17

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N =  1,  n = 5000,  r =  0,    s = 4,    L = 1000,    k = 1000


------------------------------------------------
Number of degrees of freedom          :    3
Chi-square statistic                  :    1.71
p-value of test                       :    0.64

------------------------------------------------
CPU time used                    :  00:04:22.51

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N =  1,  n = 5000,  r = 26,    s = 4,    L = 1000,    k = 1000


------------------------------------------------
Number of degrees of freedom          :    3
Chi-square statistic                  :    2.90
p-value of test                       :    0.41

------------------------------------------------
CPU time used                    :  00:04:23.59

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N =  1,  n = 80,  r = 15,    s = 15,    L = 5000,    k = 5000


------------------------------------------------
Number of degrees of freedom          :    2
Chi-square statistic                  :    0.59
p-value of test                       :    0.74

------------------------------------------------
CPU time used                    :  00:03:42.57

Generator state:



************************************************************
HOST =

GenP2


smarsa_MatrixRank test:
------------------------------------------------
   N =  1,  n = 80,  r =  0,    s = 30,    L = 5000,    k = 5000


------------------------------------------------
Number of degrees of freedom          :    2
Chi-square statistic                  :    0.66
p-value of test                       :    0.72

------------------------------------------------
CPU time used                    :  00:03:13.82

Generator state:



************************************************************
HOST =

GenP2


smarsa_Savir2 test:
------------------------------------------------
   N = 10,  n = 10000000,  r = 10,    m = 1048576,    t = 30


------------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.31
p-value of test                       :    0.13

Kolmogorov-Smirnov- statistic = D-    :   0.048
p-value of test                       :    0.93

Anderson-Darling statistic = A2       :    0.96
p-value of test                       :    0.37

Test on the sum of all N observations
Number of degrees of freedom          :  130
Chi-square statistic                  :  110.42
p-value of test                       :    0.89

------------------------------------------------
CPU time used                    :  00:02:28.07

Generator state:



************************************************************
HOST =

GenP2


smarsa_GCD test:
------------------------------------------------
   N = 10,  n = 50000000,  r =  0,   s = 30


------------------------------------------------
Test results for GCD values:

Kolmogorov-Smirnov+ statistic = D+    :    0.38
p-value of test                       :    0.04

Kolmogorov-Smirnov- statistic = D-    :   0.093
p-value of test                       :    0.79

Anderson-Darling statistic = A2       :    1.49
p-value of test                       :    0.18

Test on the sum of all N observations
Number of degrees of freedom          : 17430
Chi-square statistic                  :17267.41
p-value of test                       :    0.81


------------------------------------------------
CPU time used                    :  00:02:18.32

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
------------------------------------------------
   N =  1,  n = 100000000,  r =  0,   s = 5,   L0 =   50,   L1 =
50
```

Universitat d'Alacant
Universidad de Alicante

```
-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :   36
ChiSquare statistic                   :   34.90
p-value of test                       :    0.52


-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :   35
ChiSquare statistic                   :   26.14
p-value of test                       :    0.86


-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          :   25
ChiSquare statistic                   :    8.82
p-value of test                       :    0.9988


-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :   24
ChiSquare statistic                   :   33.59
p-value of test                       :    0.09


-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :   17
ChiSquare statistic                   :   13.38
p-value of test                       :    0.71


-----------------------------------------------
CPU time used                    :  00:01:46.76

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
-----------------------------------------------
   N =  1,  n = 100000000,  r = 25,   s = 5,   L0 =   50,   L1 =
50


-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :   36
ChiSquare statistic                   :   34.15
p-value of test                       :    0.56


-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :   35
ChiSquare statistic                   :   45.29
p-value of test                       :    0.11


-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          :   25
ChiSquare statistic                   :   25.20
p-value of test                       :    0.45


-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :   24
ChiSquare statistic                   :   25.96
p-value of test                       :    0.36


-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :   17
ChiSquare statistic                   :   13.03
p-value of test                       :    0.73


-----------------------------------------------
CPU time used                    :  00:01:46.18

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
-----------------------------------------------
   N =  1,  n = 10000000,  r =  0,   s = 10,   L0 = 1000,   L1 =
1000


-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :  146
ChiSquare statistic                   :  137.71
p-value of test                       :    0.68


-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :  146
ChiSquare statistic                   :  143.82
p-value of test                       :    0.54


-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          :  500
ChiSquare statistic                   :  530.12
p-value of test                       :    0.17


-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :  136
ChiSquare statistic                   :  113.19
p-value of test                       :    0.92


-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :   74
ChiSquare statistic                   :   87.66
p-value of test                       :    0.13


-----------------------------------------------
CPU time used                    :  00:01:52.34

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
-----------------------------------------------
   N =  1,  n = 10000000,  r = 20,   s = 10,   L0 = 1000,   L1 =
1000


-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :  146
ChiSquare statistic                   :  127.81
p-value of test                       :    0.86


-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :  146
ChiSquare statistic                   :  122.10
p-value of test                       :    0.93


-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          :  500
ChiSquare statistic                   :  516.04
p-value of test                       :    0.30


-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :  136
ChiSquare statistic                   :  164.30
p-value of test                       :    0.05


-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :   74
ChiSquare statistic                   :   68.66
p-value of test                       :    0.65


-----------------------------------------------
CPU time used                    :  00:01:51.89

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
-----------------------------------------------
   N =  1,  n = 1000000,  r =  0,   s = 15,   L0 = 10000,   L1 =
10000


-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :  384
ChiSquare statistic                   :  398.89
p-value of test                       :    0.29


-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :  384
ChiSquare statistic                   :  371.59
p-value of test                       :    0.67


-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          : 5000
ChiSquare statistic                   : 4926.44
```

Universitat d'Alacant
Universidad de Alicante

```
p-value of test                       :    0.77

-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :  378
ChiSquare statistic                   :  337.03
p-value of test                       :    0.94

-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :  200
ChiSquare statistic                   :  232.26
p-value of test                       :    0.06

-----------------------------------------------
CPU time used                    :  00:01:45.40

Generator state:



************************************************************
HOST =

GenP2


swalk_RandomWalk1 test:
-----------------------------------------------
   N =  1,  n = 1000000,  r = 15,   s = 15,   L0 = 10000,   L1 =
10000



-----------------------------------------------
Test on the values of the Statistic H

Number of degrees of freedom          :  384
ChiSquare statistic                   :  366.88
p-value of test                       :    0.73

-----------------------------------------------
Test on the values of the Statistic M

Number of degrees of freedom          :  384
ChiSquare statistic                   :  349.25
p-value of test                       :    0.90

-----------------------------------------------
Test on the values of the Statistic J

Number of degrees of freedom          : 5000
ChiSquare statistic                   : 4963.23
p-value of test                       :    0.64

-----------------------------------------------
Test on the values of the Statistic R

Number of degrees of freedom          :  378
ChiSquare statistic                   :  394.29
p-value of test                       :    0.27

-----------------------------------------------
Test on the values of the Statistic C

Number of degrees of freedom          :  200
ChiSquare statistic                   :  191.42
p-value of test                       :    0.66

-----------------------------------------------
CPU time used                    :  00:01:44.89

Generator state:



************************************************************
HOST =

GenP2


scomp_LinearComp test:
-----------------------------------------------
   N =  1,  n = 400020,  r =  0,    s = 1


-----------------------------------------------
Number of degrees of freedom          :   12
Chi2 statistic for size of jumps      :    7.33
p-value of test                       :    0.83

-----------------------------------------------
Normal statistic for number of jumps  :    1.68
p-value of test                       :    0.05


-----------------------------------------------
CPU time used                    :  00:02:52.29

Generator state:



************************************************************
HOST =

GenP2


scomp_LinearComp test:
-----------------------------------------------
   N =  1,  n = 400020,  r = 29,    s = 1


-----------------------------------------------
Number of degrees of freedom          :   12
Chi2 statistic for size of jumps      :   13.39
p-value of test                       :    0.34


-----------------------------------------------
Normal statistic for number of jumps  : 2.98e-3
p-value of test                       :    0.50


-----------------------------------------------
CPU time used                    :  00:02:52.25

Generator state:



************************************************************
HOST =

GenP2


scomp_LempelZiv test:
-----------------------------------------------
   N = 10,  n = 134217728,  r =  0,   s =   30,   k =   27


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.12
p-value of test                       :    0.70

Kolmogorov-Smirnov- statistic = D-    :    0.28
p-value of test                       :    0.18

Anderson-Darling statistic = A2       :    1.16
p-value of test                       :    0.28

Tests on the sum of all N observations
Standardized normal statistic         :    1.06
p-value of test                       :    0.14

Sample variance                       :    0.61
p-value of test                       :    0.79

-----------------------------------------------
CPU time used                    :  00:01:01.82

Generator state:



************************************************************
HOST =

GenP2


scomp_LempelZiv test:
-----------------------------------------------
   N = 10,  n = 134217728,  r = 15,   s =   15,   k =   27


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.18
p-value of test                       :    0.47

Kolmogorov-Smirnov- statistic = D-    :   0.048
p-value of test                       :    0.93

Anderson-Darling statistic = A2       :    0.46
p-value of test                       :    0.79

Tests on the sum of all N observations
Standardized normal statistic         :   -0.82
p-value of test                       :    0.79

Sample variance                       :    0.90
p-value of test                       :    0.52

-----------------------------------------------
CPU time used                    :  00:01:06.17

Generator state:



************************************************************
HOST =

GenP2


sspectral_Fourier3 test:
-----------------------------------------------
   N = 100000,  n = 16384,  r =  0,   s =    3,   k =   14


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :   0.014
p-value of test                       :    0.21

Kolmogorov-Smirnov- statistic = D-    : 5.76e-3
p-value of test                       :    0.76

Anderson-Darling statistic = A2       :    0.92
p-value of test                       :    0.40

-----------------------------------------------
CPU time used                    :  00:01:14.45

Generator state:



************************************************************
HOST =

GenP2


sspectral_Fourier3 test:
-----------------------------------------------
   N = 100000,  n = 16384,  r = 27,   s =    3,   k =   14

-----------------------------------------------
```

Universitat d'Alacant
Universidad de Alicante

```
Kolmogorov-Smirnov+ statistic = D+    :   0.012
p-value of test                       :    0.31

Kolmogorov-Smirnov- statistic = D-    : 1.58e-3
p-value of test                       :    0.98

Anderson-Darling statistic = A2       :    0.80
p-value of test                       :    0.48

-----------------------------------------------
CPU time used                    :  00:01:14.70

Generator state:



************************************************************
HOST =

GenP2


sstring_LongestHeadRun test:
-----------------------------------------------
   N =  1,  n = 1000,  r =  0,   s = 3,   L = 10000020


-----------------------------------------------
Number of degrees of freedom          :    8
Chi-square statistic                  :   15.19
p-value of test                       :    0.06

-----------------------------------------------
Global longest run of 1               :   31.00
p-value of test                       :    0.69



-----------------------------------------------
CPU time used                    :  00:03:27.93

Generator state:



************************************************************
HOST =

GenP2


sstring_LongestHeadRun test:
-----------------------------------------------
   N =  1,  n = 1000,  r = 27,   s = 3,   L = 10000020


-----------------------------------------------
Number of degrees of freedom          :    8
Chi-square statistic                  :    7.69
p-value of test                       :    0.46

-----------------------------------------------
Global longest run of 1               :   32.00
p-value of test                       :    0.50



-----------------------------------------------
CPU time used                    :  00:03:28.56

Generator state:



************************************************************
HOST =

GenP2


sstring_PeriodsInStrings test:
-----------------------------------------------
   N = 10,  n = 500000000,  r =  0,   s =   10


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.33
p-value of test                       :    0.09

Kolmogorov-Smirnov- statistic = D-    :    0.15
p-value of test                       :    0.57

Anderson-Darling statistic = A2       :    0.99
p-value of test                       :    0.36

Test on the sum of all N observations
Number of degrees of freedom          :  200
Chi-square statistic                  :  187.06
p-value of test                       :    0.74

-----------------------------------------------
CPU time used                    :  00:06:04.82

Generator state:



************************************************************
HOST =

GenP2


sstring_PeriodsInStrings test:
-----------------------------------------------
   N = 10,  n = 500000000,  r = 20,   s =   10


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.22
p-value of test                       :    0.32

Kolmogorov-Smirnov- statistic = D-    :   0.051
p-value of test                       :    0.92

Anderson-Darling statistic = A2       :    0.83
p-value of test                       :    0.46

Test on the sum of all N observations
Number of degrees of freedom          :  200
Chi-square statistic                  :  177.15
p-value of test                       :    0.88

-----------------------------------------------
CPU time used                    :  00:06:04.82

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingWeight2 test:
-----------------------------------------------
   N = 10,  n = 1000000000,  r =  0,   s = 3,   L = 1000000


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.28
p-value of test                       :    0.18

Kolmogorov-Smirnov- statistic = D-    :   0.057
p-value of test                       :    0.91

Anderson-Darling statistic = A2       :    1.12
p-value of test                       :    0.30

Test on the sum of all N observations
Number of degrees of freedom          : 10000
Chi-square statistic                  : 9821.17
p-value of test                       :    0.90

-----------------------------------------------
CPU time used                    :  00:02:52.40

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingWeight2 test:
-----------------------------------------------
   N = 10,  n = 1000000000,  r = 27,   s = 3,   L = 1000000


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.27
p-value of test                       :    0.19

Kolmogorov-Smirnov- statistic = D-    :    0.18
p-value of test                       :    0.47

Anderson-Darling statistic = A2       :    0.79
p-value of test                       :    0.48

Test on the sum of all N observations
Number of degrees of freedom          : 10000
Chi-square statistic                  : 9989.70
p-value of test                       :    0.53

-----------------------------------------------
CPU time used                    :  00:02:53.59

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingCorr test:
-----------------------------------------------
   N =  1,  n = 1000000000,  r = 10,   s = 10,   L = 30


-----------------------------------------------
Normal statistic                      :   -0.43
p-value of test                       :    0.67

-----------------------------------------------
CPU time used                    :  00:02:58.06

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingCorr test:
-----------------------------------------------
   N =  1,  n = 100000000,  r = 10,   s = 10,   L = 300


-----------------------------------------------
Normal statistic                      :   -0.76
p-value of test                       :    0.78

-----------------------------------------------
CPU time used                    :  00:02:49.92

Generator state:



************************************************************
HOST =
```

Universitat d'Alacant
Universidad de Alicante

```
GenP2


sstring_HammingCorr test:
-----------------------------------------------
   N =  1,  n = 100000000,  r = 10,   s = 10,   L = 1200


-----------------------------------------------
Normal statistic                    :   -0.20
p-value of test                     :    0.58

-----------------------------------------------
CPU time used                   :  00:11:17.65

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N = 10,  n = 30000000,  r =  0,   s = 3,   L = 30,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.082
p-value of test                     :    0.83

Kolmogorov-Smirnov- statistic = D-    :    0.23
p-value of test                     :    0.29

Anderson-Darling statistic = A2       :    0.84
p-value of test                     :    0.45

Test on the sum of all N observations
Number of degrees of freedom          : 4890
Chi-square statistic                  : 4995.99
p-value of test                     :    0.14

-----------------------------------------------
CPU time used                   :  00:05:15.28

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N = 10,  n = 30000000,  r = 27,   s = 3,   L = 30,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.13
p-value of test                     :    0.65

Kolmogorov-Smirnov- statistic = D-    :    0.26
p-value of test                     :    0.23

Anderson-Darling statistic = A2       :    0.66
p-value of test                     :    0.59

Test on the sum of all N observations
Number of degrees of freedom          : 4890
Chi-square statistic                  : 4926.97
p-value of test                     :    0.35

-----------------------------------------------
CPU time used                   :  00:05:15.46

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N =  1,  n = 30000000,  r =  0,   s = 4,   L = 300,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------
Number of degrees of freedom          : 4117
Chi-square statistic                  : 4198.48
p-value of test                     :    0.18

-----------------------------------------------
CPU time used                   :  00:03:55.31

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N =  1,  n = 30000000,  r = 26,   s = 4,   L = 300,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------
Number of degrees of freedom          : 4117
Chi-square statistic                  : 4245.98
p-value of test                     :    0.08

-----------------------------------------------
CPU time used                   :  00:03:56.00

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N =  1,  n = 10000000,  r =  0,   s = 5,   L = 1200,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------
Number of degrees of freedom          : 11825
Chi-square statistic                  :11661.77
p-value of test                     :    0.86

-----------------------------------------------
CPU time used                   :  00:04:14.00

Generator state:



************************************************************
HOST =

GenP2


sstring_HammingIndep test:
-----------------------------------------------
   N =  1,  n = 10000000,  r = 25,   s = 5,   L = 1200,   d = 0


Counters with expected numbers >= 10
-----------------------------------------------
Number of degrees of freedom          : 11825
Chi-square statistic                  :11735.65
p-value of test                     :    0.72

-----------------------------------------------
CPU time used                   :  00:04:16.46

Generator state:



************************************************************
HOST =

GenP2


sstring_Run test:
-----------------------------------------------
   N =  1,  n = 2000000000,  r =  0,   s =    3

-----------------------------------------------
Total number of 1 runs:  2000000000

Number of degrees of freedom          :   54
Chi2 statistic for number of runs     :   50.96
p-value of test                     :    0.59

-----------------------------------------------
Total number of bits:  8000076204

Normal statistic for number of bits   :    0.60
p-value of test                     :    0.27


-----------------------------------------------
CPU time used                   :  00:03:05.20

Generator state:



************************************************************
HOST =

GenP2


sstring_Run test:
-----------------------------------------------
   N =  1,  n = 2000000000,  r = 27,   s =    3

-----------------------------------------------
Total number of 1 runs:  2000000000

Number of degrees of freedom          :   54
Chi2 statistic for number of runs     :   74.16
p-value of test                     :    0.04

-----------------------------------------------
Total number of bits:  7999970349

Normal statistic for number of bits   :   -0.23
p-value of test                     :    0.59


-----------------------------------------------
CPU time used                   :  00:03:06.29

Generator state:
```

Universitat d'Alacant
Universidad de Alicante

```
***********************************************************
HOST =

GenP2


sstring_AutoCor test:
-----------------------------------------------
   N = 10,  n = 1000000030,  r =  0,   s = 3,   d = 1


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.11
p-value of test                       :    0.72

Kolmogorov-Smirnov- statistic = D-    :    0.21
p-value of test                       :    0.36

Anderson-Darling statistic = A2       :    0.56
p-value of test                       :    0.68

Tests on the sum of all N observations
Standardized normal statistic         :    0.73
p-value of test                       :    0.23

Sample variance                       :    0.75
p-value of test                       :    0.66

-----------------------------------------------
CPU time used                    :  00:04:03.35

Generator state:



***********************************************************
HOST =

GenP2


sstring_AutoCor test:
-----------------------------------------------
   N = 10,  n = 1000000029,  r =  0,   s = 3,   d = 3


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :   0.065
p-value of test                       :    0.88

Kolmogorov-Smirnov- statistic = D-    :    0.31
p-value of test                       :    0.11

Anderson-Darling statistic = A2       :    1.33
p-value of test                       :    0.22

Tests on the sum of all N observations
Standardized normal statistic         :    1.35
p-value of test                       :    0.09

Sample variance                       :    1.27
p-value of test                       :    0.25

-----------------------------------------------
CPU time used                    :  00:03:44.25

Generator state:



***********************************************************
HOST =

GenP2
```

```
sstring_AutoCor test:
-----------------------------------------------
   N = 10,  n = 1000000030,  r = 27,   s = 3,   d = 1


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.17
p-value of test                       :    0.49

Kolmogorov-Smirnov- statistic = D-    :    0.15
p-value of test                       :    0.58

Anderson-Darling statistic = A2       :    0.31
p-value of test                       :    0.93

Tests on the sum of all N observations
Standardized normal statistic         :    0.11
p-value of test                       :    0.46

Sample variance                       :    0.75
p-value of test                       :    0.67

-----------------------------------------------
CPU time used                    :  00:04:03.59

Generator state:



***********************************************************
HOST =

GenP2


sstring_AutoCor test:
-----------------------------------------------
   N = 10,  n = 1000000029,  r = 27,   s = 3,   d = 3


-----------------------------------------------

Kolmogorov-Smirnov+ statistic = D+    :    0.31
p-value of test                       :    0.12

Kolmogorov-Smirnov- statistic = D-    :    0.14
p-value of test                       :    0.63

Anderson-Darling statistic = A2       :    1.00
p-value of test                       :    0.36

Tests on the sum of all N observations
Standardized normal statistic         :   -0.71
p-value of test                       :    0.76

Sample variance                       :    1.65
p-value of test                       :    0.09

-----------------------------------------------
CPU time used                    :  00:03:44.06

Generator state:



========= Summary results of BigCrush =========

Version:
Generator:        GenP2
Number of statistics:  160
Total CPU time:   07:33:21.54

All tests were passed
```

Universitat d'Alacant
Universidad de Alicante

Ejemplo comparativo de solamente el sumario de un análisis desfavorable de Big Crush de TestU01 para una la combinación 65537 1-2-10 sin filtro *s-box*:

```
========= Summary results of BigCrush =========

Version:
Generator:        GenP2
Number of statistics:  160
Total CPU time:   07:21:58.39
The following tests gave p-values outside [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

      Test                          p-value
----------------------------------------------
28  SimpPoker, r = 0                  eps
29  SimpPoker, r = 25                 eps
30  CouponCollector, r = 0            eps
31  CouponCollector, r = 10           eps
32  CouponCollector, r = 20           eps
33  CouponCollector, r = 27           eps
34  Gap, r = 0                        eps
35  Gap, r = 25                    3.3e-12
36  Gap, r = 0                     3.2e-9
37  Gap, r = 20                       eps
50  SampleProd, t = 8              6.8e-11
51  SampleProd, t = 16             1.5e-12
52  SampleProd, t = 24             2.3e-24
53  SampleMean, r = 0               3.9e-4
54  SampleMean, r = 10              8.1e-4
59  WeightDistrib, r = 0            2.5e-9
60  WeightDistrib, r = 20           4.8e-8
61  WeightDistrib, r = 28           5.1e-7
65  SumCollector                      eps
72  Savir2                          1.3e-8
88  PeriodsInStrings, r = 0        5.2e-80
89  PeriodsInStrings, r = 20       2.1e-66
----------------------------------------------
All other tests were passed
```

Tabla de Velocidades

	d	s/Gb	bytes/Ite.	MB/s	Iteraciones/GB	s/Iteración
M7	7	3,58	46	286,20	23342213,57	1,53E-07
M8	8	3,52	52	291,29	20648881,23	1,70E-07
M9	9	3,44	58	297,26	18512790,07	1,86E-07
M10	10	3,52	64	291,29	16777216	2,10E-07
M11	11	3,48	70	294,61	15339168,91	2,27E-07
M12	12	3,52	76	291,15	14128181,89	2,49E-07
M13	13	3,63	82	281,71	13094412,49	2,78E-07
M14	14	3,68	88	277,90	12201611,64	3,02E-07
M15	15	3,55	94	288,81	11422785,36	3,10E-07
M19	19	3,51	118	291,37	9099506,983	3,86E-07

Velocidades comparativas

	AES	RC4	Salsa20	HC128
MB/s	60,04	154,18	137,67	131,19

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

Apéndice B
Códigos fuente

Los códigos fuente que se muestran en este apéndice están en lenguaje C y se han compilado para 64 bits y MS-Windows con el entorno de desarrollo Dev-C++ [69] y el compilador TDM-GCC (en concreto Dev-C++ 5.11 con TDM-GCC 4.9.2 64bit) disponible en el enlace:

http://orwelldevcpp.blogspot.com

El generador para distintos primos usado en las dos primeras baterías de pruebas se proporciona en dos ficheros, un fichero fuente en C y otro de cabecera que implementa las tablas de polinomios primitivos.

Para el generador con caja de sustitución y el generador óptimo, se proporciona la funcionalidad del propio generador en un módulo aparte del programa principal. El módulo, llamado motor, aísla de la implementación realizada, manteniendo en privado la estructura y operaciones internas del generador y facilitando, únicamente, el interfaz necesario para la obtención de la secuencia en un buffer. El programa principal sirve, además de para realizar las pruebas, como ejemplo de uso del módulo y su interfaz.

En estos dos casos, para facilitar la compilación integrada del módulo con el programa, se dispone de los correspondientes proyectos para el entorno Dev-C++, que no se muestran en este apéndice y están disponibles en los enlaces facilitados.

Universitat d'Alacant
Universidad de Alicante

B.1 Generador para distintos primos

Las instrucciones para el correcto uso y paso de parámetros, que se muestran a continuación, se obtienen al invocar el ejecutable sin parámetros:

```
Usage: MTSB extraction l option p r s t i file |
       MTSB test l p r s t

        extraction:     test (Only test matrices) |
                        slow (1 bit for iteration) | fast (1 bit for element) |
                        byte (byte by element) | double (2 bytes by element) |
                        byte0r (with half of zeros replacement) |
                        double0r (with 1/2 0s replacement)
        l:              Sequence size (or max. iterations on test)
        option:         -n (No filtered) | -f (Filtered) |
                        -u2 (Unpacked to 2 bytes) | -p4 (Packed from 4 bits)
                        -u2f (Unpacked to 2 bytes filtered) |
                        -p4f (Packed from 4 bits filtered)
        p:              Prime number
        r s t:          A B C matrices sizes
        i:              Random seed identification (call number)
        file:           Sequence output file name
```

El test a las matrices comprueba, mediante sucesivas potencias, el orden de los tres bloques matriz que se soliciten, verificando que la raíz o polinomio primitivo para el primo y tamaños escogidos es correcto.

La complejidad y extensión del código usado en las dos primeras baterías viene justificado por la polivalencia del código para distintos tipos de pruebas, y a que la importancia del coste computacional en estas pruebas es relativa porque las comparativas se realizan entre distintos tipos de filtrado, parámetros, etc., y no respecto a otro tipo de generadores.

El código mostrado a continuación puede servir de consulta rápida y aclaración sobre algún punto concreto de la implementación. Está disponible el mismo código en el siguiente enlace:

http://www.dccia.ua.es/~fmartine/tesis/MTSB.zip

MTSB.c:

```c
/*
  MTSB: Pseudorandom number generator based upon 3x3 block upper triangular
    matrix (Matrices Triangulares Superiores por Bloques de 3x3 bloques).

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/


/* headers */

#include <windows.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "primitive-tables.h"


/* data types */

typedef unsigned long long element;
typedef unsigned char byte;

typedef enum {false, true} bool;

typedef struct
{
  unsigned long l;
  element * e;
} data;

typedef enum
{
  test,
  slow,
  fast,
  oneByte,
  doubleByte
} kind;

typedef enum
{
  directed,
  unpacked2bytes,
  packed4bits
} out;

typedef enum
{
  noFiltered,
  filtered,
} filter;

typedef struct _arguments
{
  kind k;
  bool zeroReplacement;
  unsigned long l;
  out o;
  filter f;
  unsigned long p;
  unsigned r;
  unsigned s;
  unsigned t;
  unsigned i;
  char * output;
} arguments;

#define MAX_TEST 3

typedef struct _timeRegister
{
  unsigned long freq;
  unsigned long first;
```

Universitat d'Alacant
Universidad de Alicante

```
77    unsigned long last;
78    unsigned long initial;
79    unsigned short n_t;
80    unsigned long test[MAX_TEST];
81    unsigned long n_e;
82    unsigned long min_e;
83    unsigned long max_e;
84    double mean_e;
85    unsigned long n_f;
86    unsigned long min_f;
87    unsigned long max_f;
88    double mean_f;
89  } timeRegister;
90
91  typedef struct _extraction
92  {
93    kind k;
94    bool zeroReplacement;
95    bool replaceZero;
96    unsigned long l;
97    unsigned long pos;
98    byte * b;
99    filter f;
100   out o;
101   byte c;
102   byte s;
103   unsigned short remainderBits;
104 } extraction;
105
106 typedef struct _matrix
107 {
108   unsigned long prime;
109   unsigned rows;
110   unsigned cols;
111   data * d;
112 } matrix;
113
114 typedef struct _seed
115 {
116   unsigned long p;
117   unsigned r;
118   unsigned s;
119   unsigned t;
120   matrix * X;
121   matrix * Y;
122   matrix * Z;
123   element zeroReplacement;
124 } seed;
125
126 typedef struct _context
127 {
128   timeRegister * tr;
129   unsigned long l;
130   unsigned long pos;
131   unsigned long h;
132   seed * s;
133   matrix * A;
134   matrix * B;
135   matrix * C;
136   element primitiveRoot;
137   matrix * Ah;
138   matrix * Bh;
139   matrix * Xh;
140   matrix * Yh;
141   matrix * Zh;
142   extraction * e;
143   matrix * auxA;
144   matrix * auxB;
145   matrix * auxX1;
146   matrix * auxX2;
147   matrix * auxX3;
148   matrix * auxX4;
149   matrix * auxY1;
150   matrix * auxY2;
151   matrix * auxY3;
152   matrix * auxZ1;
153   matrix * auxZ2;
154 } context;
155
```

Universitat d'Alacant
Universidad de Alicante

```
156
157 /* prototypes */
158
159 void Usage(char * command);
160 arguments * GetArguments(int argc, char * argv[]);
161 void DeleteArguments(arguments * args);
162 void PrintArguments(arguments * args);
163
164 void TimeStamp();
165
166 timeRegister * CreateTimeRegister();
167 void DeleteTimeRegister(timeRegister * tr);
168 void PrintTimeRegister(timeRegister * tr);
169 void RegisterInicialization(timeRegister * tr);
170 void RegisterTest(timeRegister * tr);
171 void RegisterExtraction(timeRegister * tr);
172 void RegisterFiltration(timeRegister * tr);
173
174 seed * CreateSeed(unsigned long p, unsigned r,
175                   unsigned s, unsigned t, unsigned sr);
176 void DeleteSeed(seed * sd);
177 void PrintSeed(seed * sd);
178
179 extraction * CreateExtraction(kind k, bool zr, unsigned r, unsigned s,
180                               unsigned t, out o, filter f);
181 void DeleteExtraction(extraction * ext);
182 void PrintExtraction(extraction * ext);
183
184 context * CreateContext(arguments * arg);
185 void DeleteContext(context * ctx);
186 void PrintContext(context * ctx, char * message);
187
188 void GetExtraction(context * ctx);
189 void FiltrateExtraction(FILE * file, context * ctx);
190
191 data * CreateData(unsigned long l);
192 void DeleteData(data * d);
193
194 matrix * CreateMatrix(unsigned long prime, unsigned rows, unsigned cols);
195 void DeleteMatrix(matrix * mat);
196 void PrintMatrix(matrix * mat);
197 void FillAssociatedMatrix(matrix * dest);
198 element GetPrimitiveRoot(unsigned long p);
199 void FillPrimitiveVector(element * dest, unsigned long p, unsigned d);
200 void AddMatrices(matrix * dest, matrix * src1, matrix * src2);
201 void MultMatrices(matrix * dest, matrix * src1, matrix * src2);
202 void CopyMatrices(matrix * dest, matrix * src);
203 void ModMatrix(matrix * mat);
204 int TestMatrix(matrix * mat, unsigned long max);
205
206
207 /* code */
208
209 int main(int argc, char * argv[])
210 {
211   FILE * of;
212   int err;
213   arguments * args;
214   context * ctx;
215
216   TimeStamp();
217   args = GetArguments(argc,argv);
218   PrintArguments(args);
219
220   ctx = CreateContext(args);
221
222   if (args->k == test)
223   {
224     err=TestMatrix(ctx->A,ctx->l);
225     RegisterTest(ctx->tr);
226     err|=TestMatrix(ctx->B,ctx->l);
227     RegisterTest(ctx->tr);
228     err|=TestMatrix(ctx->C,ctx->l);
229     RegisterTest(ctx->tr);
230   }
231   else
232   {
233     err=fopen_s(&of,args->output,"wb");
234     if (err != 0)
```

```
235     {
236       printf("Error opening the output file. Error code %d.\n", err);
237       exit(EXIT_FAILURE);
238     }
239     RegisterInicialization(ctx->tr);
240     while ((ctx->l==0) || (ctx->pos < ctx->l))
241     {
242       GetExtraction(ctx);
243       RegisterExtraction(ctx->tr);
244       FiltrateExtraction(of, ctx);
245       RegisterFiltration(ctx->tr);
246     }
247     fclose(of);
248   }
249
250   PrintContext(ctx,"Context at end:");
251   if (err != EXIT_SUCCESS)
252     exit(EXIT_FAILURE);
253
254   DeleteContext(ctx);
255   TimeStamp();
256
257   return EXIT_SUCCESS;
258 }
259
260
261 void Usage(char * command)
262 {
263   char drive[_MAX_DRIVE];
264   char dir[_MAX_DIR];
265   char fname[_MAX_FNAME];
266   char ext[_MAX_EXT];
267   errno_t err;
268
269   err = _splitpath_s(command,drive,_MAX_DRIVE,dir,_MAX_DIR,
270                      fname,_MAX_FNAME,ext,_MAX_EXT);
271   if (err != 0)
272   {
273     printf("Error splitting the path. Error code %d.\n", err);
274     exit(EXIT_FAILURE);
275   }
276   printf("\nUsage: %s extraction l option p r s t i file |\n",fname);
277   printf("       %s test l p r s t\n\n",fname);
278   printf("\textraction:\ttest (Only test matrices) |\n\t\t\tslow");
279   printf(" (1 bit for iteration) | fast (1 bit for element) |\n\t\t");
280   printf("\tbyte (byte by element) | double (2 bytes by element) |\n\t\t");
281   printf("\tbyteOr (with half of zeros replacement) |\n\t\t");
282   printf("\tdoubleOr (with 1/2 Os replacement)\n");
283   printf("\tl:\t\tSequence size (or max. iterations on test)\n");
284   printf("\toption:\t\t-n (No filtered) | -f (Filtered) |\n\t\t");
285   printf("\t-u2 (Unpacked to 2 bytes) | -p4 (Packed from 4 bits)\n\t\t");
286   printf("\t-u2f (Unpacked to 2 bytes filtered) |\n\t\t");
287   printf("\t-p4f (Packed from 4 bits filtered)\n");
288   printf("\tp:\t\tPrime number\n");
289   printf("\tr s t:\t\tA B C matrices sizes\n");
290   printf("\ti:\t\tRandom seed identification (call number)\n");
291   printf("\tfile:\t\tSequence output file name\n\n");
292   exit(EXIT_FAILURE);
293 }
294
295
296 arguments * GetArguments(int argc, char * argv[])
297 {
298   arguments * args;
299   char * end;
300   int i;
301
302   if ((argc != 10) && (argc != 7)) Usage(argv[0]);
303   args = (arguments *)malloc(sizeof(arguments));
304
305   if (!strcmp(argv[i=1],"test"))
306   {
307     args->k = test;
308     if (argc != 7) Usage(argv[0]);
309   }
310   else if (argc == 7 ) Usage(argv[0]);
311   else if (!strcmp(argv[i],"slow"))
312   {
313     args->k = slow;
```

```
314       args->zeroReplacement = FALSE;
315     }
316     else if (!strcmp(argv[i],"fast"))
317     {
318        args->k = fast;
319       args->zeroReplacement = FALSE;
320     }
321     else if (!strcmp(argv[i],"byte"))
322     {
323       args->k = oneByte;
324       args->zeroReplacement = FALSE;
325     }
326     else if (!strcmp(argv[i],"double"))
327     {
328       args->k = doubleByte;
329       args->zeroReplacement = FALSE;
330     }
331     else if (!strcmp(argv[i],"byte0r"))
332     {
333       args->k = oneByte;
334       args->zeroReplacement = TRUE;
335     }
336     else if (!strcmp(argv[i],"double0r"))
337     {
338       args->k = doubleByte;
339       args->zeroReplacement = TRUE;
340     }
341     else Usage(argv[0]);
342
343     args->l = strtoul(argv[++i],&end,10);
344     if (strlen(end) > 0) Usage(argv[0]);
345
346     if (argc == 10)
347     {
348       if (!strcmp(argv[++i],"-n"))
349       {
350         args->o = directed;
351         args->f = noFiltered;
352       }
353       else if (!strcmp(argv[i],"-f"))
354       {
355         args->o = directed;
356         args->f = filtered;
357       }
358       else if (!strcmp(argv[i],"-u2"))
359       {
360         args->o = unpacked2bytes;
361         args->f = noFiltered;
362       }
363       else if (!strcmp(argv[i],"-p4"))
364       {
365         args->o = packed4bits;
366         args->f = noFiltered;
367       }
368       else if (!strcmp(argv[i],"-u2f"))
369       {
370         args->o = unpacked2bytes;
371         args->f = filtered;
372       }
373       else if (!strcmp(argv[i],"-p4f"))
374       {
375         args->o = packed4bits;
376         args->f = filtered;
377       }
378       else Usage(argv[0]);
379     }
380
381     args->p = strtoul(argv[++i],&end,10);
382     if ((args->p == 0) || strlen(end) > 0) Usage(argv[0]);
383     args->r = strtoul(argv[++i],&end,10);
384     if ((args->r == 0) || strlen(end) > 0) Usage(argv[0]);
385     args->s = strtoul(argv[++i],&end,10);
386     if ((args->s == 0) || strlen(end) > 0) Usage(argv[0]);
387     args->t = strtoul(argv[++i],&end,10);
388     if ((args->t == 0) || strlen(end) > 0) Usage(argv[0]);
389
390     if (argc == 10)
391     {
392       args->i = strtoul(argv[++i],&end,10);
```

Universitat d'Alacant
Universidad de Alicante

```
393     if ((args->i == 0) || strlen(end) > 0) Usage(argv[0]);
394     args->output = argv[++i];
395   }
396
397   return args;
398 }
399
400
401 void DeleteArguments(arguments * args)
402 {
403   free(args);
404 }
405
406
407 void PrintArguments(arguments * args)
408 {
409   printf("Arguments:\n");
410   switch (args->k)
411   {
412     case test: printf("\n\nTest associated matrices");
413       break;
414     case slow: printf("\n\nExtraction methode slow");
415       break;
416     case fast: printf("\n\nExtraction methode fast");
417       break;
418     case oneByte: printf("\n\nExtraction methode byte");
419       if (args->zeroReplacement) printf(" with half of zeros replacement");
420       break;
421     case doubleByte: printf("\n\nExtraction methode double byte");
422       if (args->zeroReplacement) printf(" with half of zeros replacement");
423   }
424   if (args->k != test)
425   {
426     printf(": %d bits ",args->l);
427     switch (args->o)
428     {
429       case unpacked2bytes: printf("unpacked to 2 bytes");
430         break;
431       case packed4bits: printf("packed from 4 bits");
432     }
433     switch (args->f)
434     {
435       case noFiltered: printf(" no filtered\n");
436         break;
437       case filtered: printf(" filtered\n");
438     }
439   }
440   else printf(": %d max. iterarions\n",args->l);
441   printf("p = %d r = %d s = %d t = %d\n",args->p,args->r,args->s,args->t);
442   if (args->k != test)
443     printf("Call identification %d. Output file: %s\n",args->i,args->output);
444   printf("\n");
445 }
446
447
448 void TimeStamp()
449 {
450   char time_buff[9],date_buff[9];
451
452   if (_strtime_s(time_buff,9) || _strdate_s(date_buff,9))
453   {
454     printf("Time/Date system error.");
455     exit(EXIT_FAILURE);
456   }
457   else
458     printf("\n%s %s\n",date_buff,time_buff);
459 }
460
461
462 timeRegister * CreateTimeRegister()
463 {
464   LARGE_INTEGER li;
465   timeRegister * tr;
466
467   tr = (timeRegister *)malloc(sizeof(timeRegister));
468   QueryPerformanceFrequency(&li);
469   tr->freq = li.QuadPart;
470   QueryPerformanceCounter(&li);
471   tr->first = li.QuadPart;
```

```
472   tr->last = li.QuadPart;
473   tr->initial = 0;
474   tr->n_t = 0;
475   tr->n_e = 0;
476   tr->min_e = ULONG_MAX;
477   tr->max_e = 0;
478   tr->mean_e = 0.0;
479   tr->n_f = 0;
480   tr->min_f = ULONG_MAX;
481   tr->max_f = 0;
482   tr->mean_f = 0.0;
483
484   return tr;
485 }
486
487
488 void DeleteTimeRegister(timeRegister * tr)
489 {
490   free(tr);
491 }
492
493
494 void PrintTimeRegister(timeRegister * tr)
495 {
496   LARGE_INTEGER li;
497   unsigned long r;
498   unsigned short i;
499
500   QueryPerformanceCounter(&li);
501   r = (unsigned long)li.QuadPart;
502   printf("Performance:       Frequency: %lu\n", tr->freq);
503   printf("Initialitation:  Time cycles: %lu Seconds: %.8lf\n",
504          tr->initial,(double)tr->initial/tr->freq);
505   if (tr->n_t > 0) for (i=0;i<tr->n_t;i++)
506     printf("Test: %hu      Time cycles: %lu Seconds: %.8lf\n",
507            i+1,tr->test[i],(double)tr->test[i]/tr->freq);
508   if (tr->n_e > 0)
509   {
510     printf("Extractions: %lu\n", tr->n_e);
511     printf("             Min. Time cycles: %lu Seconds: %.8lf\n",
512            tr->min_e, (double)tr->min_e/tr->freq);
513     printf("             Max. Time cycles: %lu Seconds: %.8lf\n",
514            tr->max_e, (double)tr->max_e/tr->freq);
515     printf("            Mean Time cycles: %.0lf Seconds: %.8lf\n",
516            tr->mean_e, tr->mean_e/tr->freq);
517     printf("Filtrations: %lu\n", tr->n_f);
518     printf("             Min. Time cycles: %lu Seconds: %.8lf\n",
519            tr->min_f, (double)tr->min_f/tr->freq);
520     printf("             Max. Time cycles: %lu Seconds: %.8lf\n",
521            tr->max_f, (double)tr->max_f/tr->freq);
522     printf("            Mean Time cycles: %.0lf Seconds: %.8lf\n",
523            tr->mean_f, tr->mean_e/tr->freq);
524     if (tr->n_e > tr->n_f)
525       printf("Last filtration: Time cycles: %lu Seconds: %.8lf\n",
526              r-tr->last, (double)(r-tr->last)/tr->freq);
527   }
528   if ((tr->n_t > 0) || (tr->n_e > 0))
529   {
530     tr->last=r;
531     printf("        Counter:       First: %15lu\n", tr->first);
532     printf("                        Last: %15lu\n", tr->last);
533     printf("        Total:   Time cycles: %15lu Seconds: %.8lf\n",
534            tr->last-tr->first,(double)(tr->last-tr->first)/tr->freq);
535   }
536 }
537
538
539 void RegisterInicialization(timeRegister * tr)
540 {
541   LARGE_INTEGER li;
542   unsigned long r;
543
544   QueryPerformanceCounter(&li);
545   r = (unsigned long)li.QuadPart;
546   tr->initial=r-tr->last;
547   tr->last=r;
548 }
549
550 void RegisterTest(timeRegister * tr)
```

```
551 {
552   LARGE_INTEGER li;
553   unsigned long r;
554
555   if (tr->n_t == MAX_TEST)
556   {
557     printf("Error: Too much testing.\n");
558     exit(EXIT_FAILURE);
559   }
560   QueryPerformanceCounter(&li);
561   r = (unsigned long)li.QuadPart-tr->last;
562   tr->last+=r;
563   tr->test[tr->n_t++]=r;
564 }
565
566
567 void RegisterExtraction(timeRegister * tr)
568 {
569   LARGE_INTEGER li;
570   unsigned long r;
571
572   QueryPerformanceCounter(&li);
573   r = (unsigned long)li.QuadPart-tr->last;
574   tr->last+=r;
575   if (r < tr->min_e) tr->min_e=r;
576   if (r > tr->max_e) tr->max_e=r;
577   tr->mean_e=((tr->mean_e*tr->n_e)+r)/++(tr->n_e);
578 }
579
580
581 void RegisterFiltration(timeRegister * tr)
582 {
583   LARGE_INTEGER li;
584   unsigned long r;
585
586   QueryPerformanceCounter(&li);
587   r = (unsigned long)li.QuadPart-tr->last;
588   tr->last+=r;
589   if (r < tr->min_f) tr->min_f=r;
590   if (r > tr->max_f) tr->max_f=r;
591   tr->mean_f=((tr->mean_f*tr->n_f)+r)/++(tr->n_f);
592 }
593
594
595 seed * CreateSeed(unsigned long p, unsigned r, unsigned s, unsigned t, unsigned
596 sr)
597 {
598   unsigned long i;
599   seed * sd;
600
601   sd = (seed *)malloc(sizeof(seed));
602   sd->p = p;
603   sd->r = r;
604   sd->s = s;
605   sd->t = t;
606   sd->X = CreateMatrix(p,r,t);
607   sd->Y = CreateMatrix(p,r,s);
608   sd->Z = CreateMatrix(p,s,t);
609
610   srand(101*sr);
611   for (i=0; i<sd->X->d->l; i++) sd->X->d->e[i] = rand() % p;
612   for (i=0; i<sd->Y->d->l; i++) sd->Y->d->e[i] = rand() % p;
613   for (i=0; i<sd->Z->d->l; i++) sd->Z->d->e[i] = rand() % p;
614   sd->zeroReplacement = (rand() % (p-1))+1;
615
616   return sd;
617 }
618
619
620 void DeleteSeed(seed * sd)
621 {
622   DeleteMatrix(sd->X);
623   DeleteMatrix(sd->Y);
624   DeleteMatrix(sd->Z);
625   free(sd);
626 }
627
628
629 void PrintSeed(seed * sd)
```

Universitat d'Alacant
Universidad de Alicante

```
630 {
631   printf("\nCombination: p=%lu r=%lu s=%lu t=%lu\n",sd->p,sd->r,sd->s,sd->t);
632   printf("X:\n");
633   PrintMatrix(sd->X);
634   printf("Y:\n");
635   PrintMatrix(sd->Y);
636   printf("Z:\n");
637   PrintMatrix(sd->Z);
638   printf("Zero replacement: %lu\n",sd->zeroReplacement);
639 }
640
641
642 extraction * CreateExtraction(kind k, bool zr, unsigned r, unsigned s, unsigned
643 t, out o, filter f)
644 {
645   extraction * ext;
646
647   ext = (extraction *)malloc(sizeof(extraction));
648   ext->k = k;
649   ext->zeroReplacement = zr;
650   ext->replaceZero = FALSE;
651   if (k == slow)
652     ext->l = 1;
653   else
654   {
655     ext->l = r*t+r*s+s*t;
656     if (k == doubleByte)
657         ext->l*=2;
658   }
659   ext->b = (byte *)malloc(ext->l*sizeof(byte));
660   for (ext->pos = 0; ext->pos < ext->l; ext->b[ext->pos++] = 0);
661   ext->o = o;
662   ext->f = f;
663   ext->c = 0;
664   ext->s = 0;
665   ext->remainderBits = 0;
666
667   return ext;
668 }
669
670
671 void DeleteExtraction(extraction * ext)
672 {
673   free(ext->b);
674   free(ext);
675 }
676
677
678 void PrintExtraction(extraction * ext)
679 {
680   unsigned long i;
681
682   printf("\nExtraction: ");
683   switch (ext->k)
684   {
685     case slow: printf("slow");
686       break;
687     case fast: printf("fast");
688       break;
689     case oneByte: printf("byte");
690       break;
691     case doubleByte: printf("double byte");
692   }
693   switch (ext->o)
694   {
695     case unpacked2bytes: printf(" unpacked to 2 bytes");
696       break;
697     case packed4bits: printf(" packed from 4 bits");
698   }
699   if (ext->f == filtered)
700     printf(" filtered\n");
701   else
702     printf("\n");
703   printf("            l=%lu pos=%lu c=%hhu s=%hhu  remainder bits: %hu\n",
704          ext->l,ext->pos,ext->c,ext->s,ext->remainderBits);
705   printf("            Zero replacement: %s\tReplace next zero: %s\n",
706          ext->zeroReplacement?"True":"False",ext->replaceZero?"True":"False");
707   for (i=0; i < ext->l; i++)
708   {
```

Universitat d'Alacant
Universidad de Alicante

```
709     printf("%hhu",((ext->b[i]) & 128) ? 1 : 0);
710               printf("%hhu",((ext->b[i]) & 64) ? 1 : 0);
711               printf("%hhu",((ext->b[i]) & 32) ? 1 : 0);
712               printf("%hhu",((ext->b[i]) & 16) ? 1 : 0);
713               printf("%hhu",((ext->b[i]) & 8) ? 1 : 0);
714               printf("%hhu",((ext->b[i]) & 4) ? 1 : 0);
715               printf("%hhu",((ext->b[i]) & 2) ? 1 : 0);
716               printf("%hhu",((ext->b[i]) & 1) ? 1 : 0);
717               if (i < ext->pos)
718       printf(" %lu\n",i);
719     else
720       printf("\n");
721   }
722   printf("\n");
723 }
724
725
726 context * CreateContext(arguments * arg)
727 {
728   context * ctx;
729
730   ctx = (context *)malloc(sizeof(context));
731
732   ctx->tr = CreateTimeRegister();
733   ctx->l = arg->l;
734   ctx->pos = 0;
735   ctx->h = 1;
736   ctx->A = CreateMatrix(arg->p,arg->r,arg->r);
737   ctx->B = CreateMatrix(arg->p,arg->s,arg->s);
738   ctx->C = CreateMatrix(arg->p,arg->t,arg->t);
739   FillAssociatedMatrix(ctx->A);
740   FillAssociatedMatrix(ctx->B);
741   FillAssociatedMatrix(ctx->C);
742
743   if (arg->k == test)
744   {
745     ctx->s = NULL;
746     ctx->Ah = NULL;
747     ctx->Bh = NULL;
748     ctx->Xh = NULL;
749     ctx->Yh = NULL;
750     ctx->Zh = NULL;
751     ctx->e = NULL;
752     ctx->auxA = NULL;
753     ctx->auxB = NULL;
754     ctx->auxX1 = NULL;
755     ctx->auxX2 = NULL;
756     ctx->auxX3 = NULL;
757     ctx->auxX4 = NULL;
758     ctx->auxY1 = NULL;
759     ctx->auxY2 = NULL;
760     ctx->auxY3 = NULL;
761     ctx->auxZ1 = NULL;
762     ctx->auxZ2 = NULL;
763   }
764   else
765   {
766     ctx->primitiveRoot = GetPrimitiveRoot(arg->p);
767     ctx->s = CreateSeed(arg->p,arg->r,arg->s,arg->t,arg->i);
768     ctx->Ah = CreateMatrix(arg->p,arg->r,arg->r);
769     ctx->Bh = CreateMatrix(arg->p,arg->s,arg->s);
770     ctx->Xh = CreateMatrix(arg->p,arg->r,arg->t);
771     ctx->Yh = CreateMatrix(arg->p,arg->r,arg->s);
772     ctx->Zh = CreateMatrix(arg->p,arg->s,arg->t);
773     ctx->e = CreateExtraction(arg->k,arg->zeroReplacement,
774                               arg->r,arg->s,arg->t,arg->o,arg->f);
775     ctx->auxA = CreateMatrix(arg->p,arg->r,arg->r);
776     ctx->auxB = CreateMatrix(arg->p,arg->s,arg->s);
777     ctx->auxX1 = CreateMatrix(arg->p,arg->r,arg->t);
778     ctx->auxX2 = CreateMatrix(arg->p,arg->r,arg->t);
779     ctx->auxX3 = CreateMatrix(arg->p,arg->r,arg->t);
780     ctx->auxX4 = CreateMatrix(arg->p,arg->r,arg->t);
781     ctx->auxY1 = CreateMatrix(arg->p,arg->r,arg->s);
782     ctx->auxY2 = CreateMatrix(arg->p,arg->r,arg->s);
783     ctx->auxY3 = CreateMatrix(arg->p,arg->r,arg->s);
784     ctx->auxZ1 = CreateMatrix(arg->p,arg->s,arg->t);
785     ctx->auxZ2 = CreateMatrix(arg->p,arg->s,arg->t);
786     CopyMatrices(ctx->Ah,ctx->A);
787     CopyMatrices(ctx->Bh,ctx->B);
```

Universitat d'Alacant
Universidad de Alicante

```
788       CopyMatrices(ctx->Xh,ctx->s->X);
789       CopyMatrices(ctx->Yh,ctx->s->Y);
790       CopyMatrices(ctx->Zh,ctx->s->Z);
791     }
792
793     return ctx;
794   }
795
796
797   void DeleteContext(context * ctx)
798   {
799     DeleteTimeRegister(ctx->tr);
800     DeleteMatrix(ctx->A);
801     DeleteMatrix(ctx->B);
802     DeleteMatrix(ctx->C);
803     if (ctx->s != NULL)
804     {
805       DeleteSeed(ctx->s);
806       DeleteMatrix(ctx->Ah);
807       DeleteMatrix(ctx->Bh);
808       DeleteMatrix(ctx->Xh);
809       DeleteMatrix(ctx->Yh);
810       DeleteMatrix(ctx->Zh);
811       DeleteExtraction(ctx->e);
812       DeleteMatrix(ctx->auxA);
813       DeleteMatrix(ctx->auxB);
814       DeleteMatrix(ctx->auxX1);
815       DeleteMatrix(ctx->auxX2);
816       DeleteMatrix(ctx->auxX3);
817       DeleteMatrix(ctx->auxX4);
818       DeleteMatrix(ctx->auxY1);
819       DeleteMatrix(ctx->auxY2);
820       DeleteMatrix(ctx->auxY3);
821       DeleteMatrix(ctx->auxZ1);
822       DeleteMatrix(ctx->auxZ2);
823     }
824     free(ctx);
825   }
826
827
828   void PrintContext(context * ctx, char * message)
829   {
830     printf("%s\n\n",message);
831     PrintTimeRegister(ctx->tr);
832     printf("Context: l=%lu pos=%lu h=%lu\n",ctx->l,ctx->pos,ctx->h);
833     printf("A=\n");
834     PrintMatrix(ctx->A);
835     printf("B=\n");
836     PrintMatrix(ctx->B);
837     printf("C=\n");
838     PrintMatrix(ctx->C);
839     if (ctx->s != NULL)
840     {
841       printf("Primitive Root: %lu\n",ctx->primitiveRoot);
842       PrintSeed(ctx->s);
843       printf("Ah=\n");
844       PrintMatrix(ctx->Ah);
845       printf("Bh=\n");
846       PrintMatrix(ctx->Bh);
847       printf("Xh=\n");
848       PrintMatrix(ctx->Xh);
849       printf("Yh=\n");
850       PrintMatrix(ctx->Yh);
851       printf("Zh=\n");
852       PrintMatrix(ctx->Zh);
853       PrintExtraction(ctx->e);
854     }
855   }
856
857
858   void GetExtraction(context * ctx)
859   {
860     unsigned long i;
861
862     if (ctx->e->pos != ctx->e->l)
863     {
864       printf("Error: Bad position (pos=%d) to begin extraction.\n",ctx->e->pos);
865       exit(EXIT_FAILURE);
866     }
```

Universitat d'Alacant
Universidad de Alicante

```
867   else
868   {
869     /* Almacenamiento provisional de Yh-1 en auxY3 */
870     CopyMatrices(ctx->auxY3,ctx->Yh);
871
872     /* Calculo de:  Yh = (Ah-1 · Y) + (Yh-1 · B) */
873     /*  como        Yh  =   auxY1   +    auxY2   */
874     MultMatrices(ctx->auxY1,ctx->Ah,ctx->s->Y);
875     MultMatrices(ctx->auxY2,ctx->Yh,ctx->B);
876     AddMatrices(ctx->Yh,ctx->auxY1,ctx->auxY2);
877     ModMatrix(ctx->Yh);
878
879     /* Calculo de:  Xh = (Ah-1 · X) + (Yh-1 · Z) + (Xh-1 · C)  */
880     /*  como        Xh = (Ah-1 · X) + (auxY3 · Z) + (Xh-1 · C) */
881     /*              Xh  =  ( auxX1   +    auxX2 )  +   auxX3   */
882     /*              Xh  =         auxX4        +     auxX3     */
883     MultMatrices(ctx->auxX1,ctx->Ah,ctx->s->X);
884     MultMatrices(ctx->auxX2,ctx->auxY3,ctx->s->Z);
885     MultMatrices(ctx->auxX3,ctx->Xh,ctx->C);
886     AddMatrices(ctx->auxX4,ctx->auxX1,ctx->auxX2);
887     AddMatrices(ctx->Xh,ctx->auxX4,ctx->auxX3);
888     ModMatrix(ctx->Xh);
889
890     /* Calculo de:  Zh = (Bh-1 · Z) + (Zh-1 · C) */
891     /*  como        Zh  =  auxZ1   +   auxZ2     */
892     MultMatrices(ctx->auxZ1,ctx->Bh,ctx->s->Z);
893     MultMatrices(ctx->auxZ2,ctx->Zh,ctx->C);
894     AddMatrices(ctx->Zh,ctx->auxZ1,ctx->auxZ2);
895     ModMatrix(ctx->Zh);
896
897     /* Calculo de:  Ah = (Ah-1 · A) */
898     /*  como        Ah  =   auxA    */
899     MultMatrices(ctx->auxA,ctx->Ah,ctx->A);
900     CopyMatrices(ctx->Ah,ctx->auxA);
901     ModMatrix(ctx->Ah);
902
903     /* Calculo de:  Bh = (Bh-1 · B) */
904     /*  como        Bh  =   auxB    */
905     MultMatrices(ctx->auxB,ctx->Bh,ctx->B);
906     CopyMatrices(ctx->Bh,ctx->auxB);
907     ModMatrix(ctx->Bh);
908
909     ctx->h++;
910     ctx->e->pos=0;
911
912     switch(ctx->e->k)
913     {
914       case slow:
915         ctx->e->b[ctx->e->pos] = 0;
916         for (i=0; i < ctx->Xh->d->l; i++)
917           ctx->e->b[ctx->e->pos] ^= (byte)(1 & ctx->Xh->d->e[i]);
918         for (i=0; i < ctx->Yh->d->l; i++)
919           ctx->e->b[ctx->e->pos] ^= (byte)(1 & ctx->Yh->d->e[i]);
920         for (i=0; i < ctx->Zh->d->l; i++)
921           ctx->e->b[ctx->e->pos] ^= (byte)(1 & ctx->Zh->d->e[i]);
922         ctx->e->pos++;
923         break;
924       case fast:
925         for (i=0; i < ctx->Xh->d->l; i++)
926           ctx->e->b[ctx->e->pos++] = (byte)(1 & ctx->Xh->d->e[i]);
927         for (i=0; i < ctx->Yh->d->l; i++)
928           ctx->e->b[ctx->e->pos++] = (byte)(1 & ctx->Yh->d->e[i]);
929         for (i=0; i < ctx->Zh->d->l; i++)
930           ctx->e->b[ctx->e->pos++] = (byte)(1 & ctx->Zh->d->e[i]);
931         break;
932       case oneByte:
933         if (ctx->e->zeroReplacement)
934         {
935           for (i=0; i < ctx->Xh->d->l; i++)
936           {
937             ctx->e->b[ctx->e->pos] = (byte)(255 & ctx->Xh->d->e[i]);
938             if (ctx->e->b[ctx->e->pos])
939               ctx->e->pos++;
940             else if (ctx->e->replaceZero)
941             {
942               ctx->s->zeroReplacement*=ctx->primitiveRoot;
943               ctx->s->zeroReplacement%=ctx->s->p;
944               ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->s->zeroReplacement);
945               ctx->e->replaceZero = false;
```

```
946             }
947             else
948             {
949               ctx->e->pos++;
950               ctx->e->replaceZero = true;
951             }
952           }
953           for (i=0; i < ctx->Yh->d->l; i++)
954           {
955             ctx->e->b[ctx->e->pos] = (byte)(255 & ctx->Yh->d->e[i]);
956             if (ctx->e->b[ctx->e->pos])
957               ctx->e->pos++;
958             else if (ctx->e->replaceZero)
959             {
960               ctx->s->zeroReplacement*=ctx->primitiveRoot;
961               ctx->s->zeroReplacement%=ctx->s->p;
962               ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->s->zeroReplacement);
963               ctx->e->replaceZero = false;
964             }
965             else
966             {
967               ctx->e->pos++;
968               ctx->e->replaceZero = true;
969             }
970           }
971           for (i=0; i < ctx->Zh->d->l; i++)
972           {
973             ctx->e->b[ctx->e->pos] = (byte)(255 & ctx->Zh->d->e[i]);
974             if (ctx->e->b[ctx->e->pos])
975               ctx->e->pos++;
976             else if (ctx->e->replaceZero)
977             {
978               ctx->s->zeroReplacement*=ctx->primitiveRoot;
979               ctx->s->zeroReplacement%=ctx->s->p;
980               ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->s->zeroReplacement);
981               ctx->e->replaceZero = false;
982             }
983             else
984             {
985               ctx->e->pos++;
986               ctx->e->replaceZero = true;
987             }
988           }
989         }
990         else
991         {
992           for (i=0; i < ctx->Xh->d->l; i++)
993             ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->Xh->d->e[i]);
994           for (i=0; i < ctx->Yh->d->l; i++)
995             ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->Yh->d->e[i]);
996           for (i=0; i < ctx->Zh->d->l; i++)
997             ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->Zh->d->e[i]);
998         }
999         break;
1000      case doubleByte:
1001        if (ctx->e->zeroReplacement)
1002        {
1003          for (i=0; i < ctx->Xh->d->l; i++)
1004          {
1005            ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Xh->d->e[i])>>8));
1006            ctx->e->b[ctx->e->pos] = (byte)(255 & (ctx->Xh->d->e[i]));
1007            if ((ctx->e->b[ctx->e->pos-1]) || (ctx->e->b[ctx->e->pos]))
1008              ctx->e->pos++;
1009            else if (ctx->e->replaceZero)
1010            {
1011              ctx->s->zeroReplacement *= ctx->primitiveRoot;
1012              ctx->s->zeroReplacement %= ctx->s->p;
1013              ctx->e->b[ctx->e->pos-1] =
1014                (byte)(255 & ((ctx->s->zeroReplacement)>>8));
1015              ctx->e->b[ctx->e->pos++] =
1016                (byte)(255 & ctx->s->zeroReplacement);
1017              ctx->e->replaceZero = false;
1018            }
1019            else
1020            {
1021              ctx->e->pos++;
1022              ctx->e->replaceZero = true;
1023            }
1024          }
```

Universitat d'Alacant
Universidad de Alicante

```
1025            for (i=0; i < ctx->Yh->d->l; i++)
1026            {
1027              ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Yh->d->e[i])>>8));
1028              ctx->e->b[ctx->e->pos] = (byte)(255 & (ctx->Yh->d->e[i]));
1029              if ((ctx->e->b[ctx->e->pos-1]) || (ctx->e->b[ctx->e->pos]))
1030                ctx->e->pos++;
1031              else if (ctx->e->replaceZero)
1032              {
1033                ctx->s->zeroReplacement *= ctx->primitiveRoot;
1034                ctx->s->zeroReplacement %= ctx->s->p;
1035                ctx->e->b[ctx->e->pos-1] =
1036                  (byte)(255 & ((ctx->s->zeroReplacement)>>8));
1037                ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->s->zeroReplacement);
1038                ctx->e->replaceZero = false;
1039              }
1040              else
1041              {
1042                ctx->e->pos++;
1043                ctx->e->replaceZero = true;
1044              }
1045            }
1046            for (i=0; i < ctx->Zh->d->l; i++)
1047            {
1048              ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Zh->d->e[i])>>8));
1049              ctx->e->b[ctx->e->pos] = (byte)(255 & (ctx->Zh->d->e[i]));
1050              if ((ctx->e->b[ctx->e->pos-1]) || (ctx->e->b[ctx->e->pos]))
1051                ctx->e->pos++;
1052              else if (ctx->e->replaceZero)
1053              {
1054                ctx->s->zeroReplacement *= ctx->primitiveRoot;
1055                ctx->s->zeroReplacement %= ctx->s->p;
1056                ctx->e->b[ctx->e->pos-1] =
1057                  (byte)(255 & ((ctx->s->zeroReplacement)>>8));
1058                ctx->e->b[ctx->e->pos++] = (byte)(255 & ctx->s->zeroReplacement);
1059                ctx->e->replaceZero = false;
1060              }
1061              else
1062              {
1063                ctx->e->pos++;
1064                ctx->e->replaceZero = true;
1065              }
1066            }
1067          }
1068          else
1069          {
1070            for (i=0; i < ctx->Xh->d->l; i++)
1071            {
1072              ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Xh->d->e[i])>>8));
1073              ctx->e->b[ctx->e->pos++] = (byte)(255 & (ctx->Xh->d->e[i]));
1074            }
1075            for (i=0; i < ctx->Yh->d->l; i++)
1076            {
1077              ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Yh->d->e[i])>>8));
1078              ctx->e->b[ctx->e->pos++] = (byte)(255 & (ctx->Yh->d->e[i]));
1079            }
1080            for (i=0; i < ctx->Zh->d->l; i++)
1081            {
1082              ctx->e->b[ctx->e->pos++] = (byte)(255 & ((ctx->Zh->d->e[i])>>8));
1083              ctx->e->b[ctx->e->pos++] = (byte)(255 & (ctx->Zh->d->e[i]));
1084            }
1085          }
1086      }
1087    }
1088
1089    if (ctx->e->pos != ctx->e->l)
1090    {
1091      printf("Error: Bad position (pos=%d) to end extraction.\n",ctx->e->pos);
1092      exit(EXIT_FAILURE);
1093    }
1094    else
1095      ctx->e->pos=0;
1096  }
1097
1098
1099  void FiltrateExtraction(FILE * file, context * ctx)
1100  {
1101    if (ctx->e->pos != 0)
1102    {
1103      printf("Error: Bad position (pos=%d) to begin filtrate.\n",ctx->e->pos);
```

Universitat d'Alacant
Universidad de Alicante

```
1104      exit(EXIT_FAILURE);
1105    }
1106    while ((ctx->e->pos < ctx->e->l) && ((ctx->l == 0) || (ctx->pos < ctx->l)))
1107    {
1108      if (ctx->e->remainderBits == 0)
1109        ctx->e->c = ctx->e->b[ctx->e->pos++];
1110      if (ctx->e->o == packed4bits)
1111        if (ctx->e->pos < ctx->e->l)
1112        {
1113          ctx->e->c = ((ctx->e->c) << 4) | (15 & (ctx->e->b[ctx->e->pos++]));
1114          ctx->e->remainderBits=0;
1115        }
1116        else
1117        {
1118          ctx->e->remainderBits=4;
1119          return;
1120        }
1121      if (ctx->e->o == unpacked2bytes)
1122      {
1123        if (ctx->e->f == filtered)
1124          ctx->e->s ^= (2 & ctx->e->c)?1:0;
1125        else
1126          ctx->e->s = (2 & ctx->e->c)?1:0;
1127        if (EOF == fputc(ctx->e->s,file))
1128        {
1129          printf("Error writing the output file.");
1130          exit(EXIT_FAILURE);
1131        }
1132        ctx->pos++;
1133        ctx->e->c = (1 & ctx->e->c)?1:0;
1134      }
1135      if (ctx->e->f == filtered)
1136        ctx->e->s ^= ctx->e->c;
1137      else
1138        ctx->e->s = ctx->e->c;
1139      if (EOF == fputc(ctx->e->s,file))
1140      {
1141        printf("Error writing the output file.");
1142        exit(EXIT_FAILURE);
1143      }
1144      if ((ctx->e->k == slow) ||
1145          (ctx->e->k == fast) ||
1146          (ctx->e->o == unpacked2bytes))
1147        ctx->pos++;
1148      else
1149        ctx->pos+=8;
1150    }
1151  }
1152
1153
1154  matrix * CreateMatrix(unsigned long prime, unsigned rows, unsigned cols)
1155  {
1156    matrix * m;
1157
1158    m = (matrix *)malloc(sizeof(matrix));
1159    m->prime = prime;
1160    m->rows = rows;
1161    m->cols = cols;
1162    m->d = CreateData(rows*cols);
1163
1164    return m;
1165  }
1166
1167
1168  void DeleteMatrix(matrix *mat)
1169  {
1170    DeleteData(mat->d);
1171    free(mat);
1172  }
1173
1174
1175  void PrintMatrix(matrix *mat)
1176  {
1177    unsigned r,c;
1178    unsigned long offs;
1179
1180    offs = 0;
1181    printf("\n");
1182    for(r=0; r<mat->rows; r++)
```

Universitat d'Alacant
Universidad de Alicante

```
1183    {
1184      printf("[ ");
1185      for(c=0; c<mat->cols; c++)
1186      {
1187        printf("\t%5llu ",mat->d->e[offs++]);
1188      }
1189      printf("]\n");
1190    }
1191    printf("\n");
1192  }
1193
1194
1195  void FillAssociatedMatrix(matrix * dest)
1196  {
1197    unsigned int i, j;
1198
1199    for (i=0; i<dest->rows; i++)
1200      for (j=0; j<dest->cols; j++)
1201        if (j==i+1)
1202          dest->d->e[i*dest->cols+j]=1;
1203        else
1204          dest->d->e[i*dest->cols+j]=0;
1205    FillPrimitiveVector(&(dest->d->e[(dest->rows-1)*dest->cols]),
1206                        dest->prime,dest->cols);
1207  }
1208
1209  element GetPrimitiveRoot(unsigned long p)
1210  {
1211    unsigned i;
1212
1213    for (i=0; (i<PRIMES_TABLE_SIZE) && (PrimesTable[i]<p); i++);
1214    if ((i==PRIMES_TABLE_SIZE) || (PrimesTable[i]>p))
1215    {
1216      printf("Error: Prime p=%lu not found to get a primitive root.\n\n",p);
1217      exit(EXIT_FAILURE);
1218    }
1219    return PrimitiveRootsTable[i];
1220  }
1221
1222
1223  void FillPrimitiveVector(element * dest, unsigned long p, unsigned d)
1224  {
1225    unsigned i, j;
1226
1227    for (i=0; (i<PRIMES_TABLE_SIZE) && (PrimesTable[i]<p); i++);
1228    if ((i==PRIMES_TABLE_SIZE) || (PrimesTable[i]>p))
1229    {
1230      printf("Error: Prime p=%lu not found to get a primitive polynomial.\n\n",p);
1231      exit(EXIT_FAILURE);
1232    }
1233    if (d==1)
1234    {
1235      dest[0]=PrimitiveRootsTable[i];
1236      return;
1237    }
1238    for (j=0;
1239         (j<MAX_TRINOMIALS) &&
1240         (PrimitiveTrinomialsTable[i][j].degree<d) &&
1241         (PrimitiveTrinomialsTable[i][j].degree>0);
1242         j++);
1243    if ((j==MAX_TRINOMIALS) ||
1244        (PrimitiveTrinomialsTable[i][j].degree>d) ||
1245        (PrimitiveTrinomialsTable[i][j].degree==0))
1246    {
1247      for (i=0;
1248           (i<POLYNOMIALS_TABLE_SIZE) &&
1249           ((PrimitivePolynomialsTable[i].prime<p) ||
1250            ((PrimitivePolynomialsTable[i].prime==p) &&
1251             (PrimitivePolynomialsTable[i].degree<d)));
1252           i++);
1253      if ((i==POLYNOMIALS_TABLE_SIZE) ||
1254          (PrimitivePolynomialsTable[i].prime>p) ||
1255          (PrimitivePolynomialsTable[i].degree>d))
1256      {
1257        printf("Error: Primitive polynomial for p=%lu and d=%u not found.\n\n",
1258               p,d);
1259        exit(EXIT_FAILURE);
1260      }
1261      for (j=0;
```

```
1262          (j<MAX_TERMS && PrimitivePolynomialsTable[i].terms[j].coefficient>0);
1263          j++)
1264       if (PrimitivePolynomialsTable[i].terms[j].coefficient>0)
1265         dest[PrimitivePolynomialsTable[i].terms[j].degree] =
1266           p - PrimitivePolynomialsTable[i].terms[j].coefficient;
1267       dest[0] = p - PrimitivePolynomialsTable[i].constantTerm;
1268     return;
1269   }
1270   dest[PrimitiveTrinomialsTable[i][j].aTerm.degree] =
1271     p - PrimitiveTrinomialsTable[i][j].aTerm.coefficient;
1272   dest[0] = p - PrimitiveTrinomialsTable[i][j].constantTerm;
1273 }
1274
1275
1276 void AddMatrices(matrix * dest, matrix * src1, matrix * src2)
1277 {
1278   unsigned long i;
1279
1280   if((src1->rows!=src2->rows)||(src2->rows!=dest->rows) ||
1281      (src1->cols!=src2->cols)||(src2->cols!=dest->cols))
1282   {
1283     printf("Error: Bad sizes in [%u X %u] and [%u X %u] to [%u X %u].\n\n",
1284             src1->rows,src1->cols,src2->rows,src2->cols,dest->rows,dest->cols);
1285     exit(EXIT_FAILURE);
1286   }
1287   for(i=0; i<(dest->d->l); i++)
1288     dest->d->e[i] = (src1->d->e[i] + src2->d->e[i]);
1289 }
1290
1291
1292 void MultMatrices(matrix * dest, matrix * src1, matrix * src2)
1293 {
1294   unsigned r,c,i;
1295   unsigned long off1,off2,offd;
1296   element v;
1297
1298   if((src1->cols!=src2->rows) ||
1299      (dest->cols!=src2->cols) ||
1300      (dest->rows!=src1->rows))
1301   {
1302     printf("Error: Bad sizes [%u X %u] by [%u X %u] to [%u X %u].\n\n",
1303             src1->rows,src1->cols,src2->rows,src2->cols,dest->rows,dest->cols);
1304     exit(EXIT_FAILURE);
1305   }
1306   offd = 0;
1307   off1 = 0;
1308   for(r=0; r<dest->rows; r++)
1309   {
1310     for(c=0; c<dest->cols; c++)
1311     {
1312       off2 = c;
1313       v = 0;
1314       for(i=0; i < (src1->cols); i++)
1315       {
1316         v += src1->d->e[off1+i] * src2->d->e[off2];
1317         off2 += src2->cols;
1318       }
1319       dest->d->e[offd+c] = v;
1320     }
1321     offd += dest->cols;
1322     off1 += src1->cols;
1323   }
1324 }
1325
1326
1327 void CopyMatrices(matrix * dest, matrix * src)
1328 {
1329   if((src->rows!=dest->rows)||(src->cols!=dest->cols))
1330   {
1331     printf("Error: Bad matrix sizes in copy [%u X %u] to [%u X %u].\n\n",
1332             src->rows,src->cols,dest->rows,dest->cols);
1333     exit(EXIT_FAILURE);
1334   }
1335   memcpy(dest->d->e,src->d->e,src->d->l*sizeof(element));
1336 }
1337
1338
1339 void ModMatrix(matrix * mat)
1340 {
```

Universitat d'Alacant
Universidad de Alicante

```
1341   unsigned long i;
1342
1343   for(i=0; i<mat->d->l; i++)
1344     mat->d->e[i] = mat->d->e[i] % mat->prime;
1345 }
1346
1347
1348 int TestMatrix(matrix * mat, unsigned long max)
1349 {
1350   unsigned long i, top;
1351   matrix * T;
1352   matrix * power;
1353
1354   T = CreateMatrix(mat->prime,mat->rows,mat->cols);
1355   power = CreateMatrix(mat->prime,mat->rows,mat->cols);
1356   top=mat->prime;
1357   for (i=1;i<mat->rows;i++) top*=mat->prime;
1358   MultMatrices(power,mat,mat);
1359   for(i=1; memcmp(mat->d->e,power->d->e,mat->d->l*sizeof(element)); i++) {
1360     MultMatrices(T,mat,power);
1361     if ((0==memcmp(T->d->e,power->d->e,T->d->l*sizeof(element)))
1362        || (i == top) || (i == max))
1363       break;
1364     CopyMatrices(power,T);
1365     ModMatrix(power);
1366   }
1367   DeleteMatrix(T);
1368   DeleteMatrix(power);
1369   if ((i != top-1) && (i != max))
1370   {
1371     printf("Error: Deficient associated matrix for p=%d and d=%d\n",
1372            mat->prime,mat->cols);
1373     printf("       Expected period: %lu\n",top-1);
1374     printf("       Checked period: %lu\n\n",i);
1375     return EXIT_FAILURE;
1376   }
1377   if (i == top-1)
1378     printf("Order of matrix for p=%d and d=%d is n=%d\n\n",
1379            mat->prime,mat->cols,i);
1380   else
1381     printf("Order of matrix for p=%d and d=%d is greater than %d\n\n",
1382            mat->prime,mat->cols,i);
1383   return EXIT_SUCCESS;
1384 }
1385
1386
1387 data * CreateData(unsigned long l)
1388 {
1389   data * dat;
1390
1391   dat = (data *)malloc(sizeof(data));
1392   dat->l = l;
1393   dat->e = (element *)malloc(l*sizeof(element));
1394
1395   return dat;
1396 }
1397
1398
1399 void DeleteData(data *dat)
1400 {
1401   if(dat->e)
1402     free(dat->e);
1403   free(dat);
1404 }
```

Universitat d'Alacant
Universidad de Alicante

primitive-tables.h:

```
/*
  PRIMITIVE-TABLES: Header of types and tables of
    prime numbres, primitive polynomials and primitive roots.

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/

#ifndef _PRIMITIVE_TABLES_H
#define _PRIMITIVE_TABLES_H

#define PRIMES_TABLE_SIZE 67
#define MAX_TRINOMIALS 6
#define POLYNOMIALS_TABLE_SIZE 33
#define MAX_TERMS 7

/* Término. Ejemplo {8,2} -> 8x^2 */
typedef struct _term
{
  unsigned long coefficient;
  unsigned char degree;
} term;

/* Trinomio. Ejemplo {5,{7,3},2} -> x^5+7x^3+2 */
typedef struct _trinomial
{
  unsigned char degree; /* El coeficiente del término de ese grado siempre 1*/
  term aTerm;    /* Término distinto al de de mayor grado y al independiente*/
  unsigned long constantTerm; /* Coeficiente término independiente (grado 0)*/
} trinomial;

/* Polinomio. Ejemplo {5,59,{{1,3},{2,2},{1,1},{0,0},{0,0},{0,0},{0,0}},3}  */
/*                                -> (para el primo 5) x^59+x^3+2x^2+x+3 */
typedef struct _polynomial
{
  unsigned long prime;
  unsigned char degree; /* El coeficiente del término de ese grado siempre 1*/
  term terms[MAX_TERMS]; /* Otros términos (no mayor grado ni independiente)*/
  unsigned long constantTerm; /* Coeficiente término independiente (grado 0)*/
  } polynomial;

unsigned long PrimesTable [PRIMES_TABLE_SIZE] =
/*   1      2      3      4      5      6      7      8      9     10      */
{    2,     3,     5,     7,    11,    13,    17,    19,    23,    29, /*10*/
     31,    37,    41,    43,    47,    53,    59,    61,    67,    71, /*20*/
     73,    79,    83,    89,    97,   101,   103,   107,   109,   113, /*30*/
    127,   131,   137,   139,   149,   151,   157,   163,   167,   173, /*40*/
    179,   181,   191,   193,   197,   199,   211,   223,   227,   229, /*50*/
    233,   239,   241,   251,   257,   263,   269,   271,   277,   281, /*60*/
    283,   293,   307,   311,   313,  3187, 65537};

unsigned long PrimitiveRootsTable [PRIMES_TABLE_SIZE] =
/*   1      2      3      4      5      6      7      8      9     10      */
{    1,     2,     2,     3,     2,     2,     3,     2,     5,     2, /*10*/
     3,     2,     6,     3,     5,     2,     2,     2,     7,     2, /*20*/
     5,     3,     2,     3,     5,     2,     5,     2,     6,     3, /*30*/
     3,     2,     3,     2,     2,     6,     5,     2,     5,     2, /*40*/
     2,     2,    19,     5,     2,     3,     2,     3,     2,     6, /*50*/
     3,     7,     7,     6,     3,     5,     2,     6,     5,     3, /*60*/
     3,     2,     5,    17,    10,     2,     3};

/*
  Thanks to PIPS, the Primitive and Irreducible Polynomial Server,
  used to calculate primitive polynomials of these tables. Link:
  http://zenfact.sourceforge.net/defaultind_COUNT_1_FILE0_PIPS.html

  And thanks to Sean Erik O'Connor and his Primpoly Version 11.0 -
  A Program for Computing Primitive Polynomials, used to calculate
  big order and prime primitive polynomials too for these tables.
  Link: http://www.seanerikoconnor.freeservers.com/Mathematics/...
            .../AbstractAlgebra/PrimitivePolynomials/overview.html
*/

trinomial PrimitiveTrinomialsTable [PRIMES_TABLE_SIZE][MAX_TRINOMIALS] =
```

Universitat d'Alacant
Universidad de Alicante

```
{
/*    2*/{{ 3, {   1, 1},   1}, { 4, {   1, 1},   1}, { 5, {   1, 2},   1},
            { 6, {   1, 1},   1}, { 7, {   1, 4},   1}, { 9, {   1, 4},   1}},
/*    3*/{{ 4, {   1, 3},   2}, { 5, {   2, 1},   1}, { 6, {   2, 1},   2},
            { 7, {   2, 2},   1}, { 8, {   1, 5},   2}, { 9, {   2, 4},   1}},
/*    5*/{{ 2, {   1, 2},   2}, { 3, {   3, 1},   2}, { 5, {   4, 1},   2},
            { 6, {   2, 1},   3}, { 7, {   4, 3},   3}, { 9, {   3, 4},   2}},
/*    7*/{{ 2, {   4, 1},   5}, { 3, {   6, 2},   4}, { 5, {   5, 3},   4},
            { 7, {   2, 5},   2}, { 8, {   2, 1},   3}, { 9, {   3, 7},   2}},
/*   11*/{{ 2, {   6, 1},   2}, { 3, {   3, 1},   9}, { 4, {   4, 3},   2},
            { 5, {   2, 2},   9}, { 7, {   2, 1},   5}, { 9, {   2, 8},   3}},
/*   13*/{{ 2, {   6, 1},   2}, { 3, {   3, 2},   2}, { 5, {   8, 3},   6},
            { 7, {   3, 1},   6}, { 9, {   4, 8},   7},                 {0,{0,0},0}},
/*   17*/{{ 2, {   1, 1},   3}, { 3, {   1, 1},   3}, { 4, {  15, 1},   6},
            { 5, {   1, 1},   3}, { 6, {   3, 5},  11}, { 7, {  13, 1},   6}},
/*   19*/{{ 2, {  10, 1},   3}, { 3, {   5, 2},   9}, { 4, {   9, 1},  10},
            { 5, {   8, 4},  16}, { 6, {  10, 1},   2}, { 7, {  14, 2},  16}},
/*   23*/{{ 2, {   3, 1},  11}, { 3, {   7, 1},   4}, { 4, {  15, 1},  15},
            { 5, {   6, 4},  12}, { 6, {  17, 5},  15}, { 7, {  13, 5},  12}},
/*   29*/{{ 2, {  12, 1},  21}, { 3, {  17, 2},  18}, { 4, {   6, 3},  10},
            { 5, {   2, 3},  27}, { 6, {   8, 5},  27}, { 7, {  13, 5},  18}},
/*   31*/{{ 2, {  28, 1},  12}, { 3, {  13, 1},  20}, { 4, {  11, 1},  21},
            { 5, {  13, 4},  18}, { 6, {  15, 5},  22}, { 7, {  26, 5},  10}},
/*   37*/{{ 2, {  31, 1},  15}, { 3, {  28, 1},  32}, { 4, {  11, 1},   2},
            { 5, {  24, 3},   5}, { 6, {  13, 1},  22}, { 7, {  25, 2},  32}},
/*   41*/{{ 2, {  24, 1},  15}, { 3, {   7, 2},   7}, { 4, {  18, 1},  12},
            { 5, {  11, 2},  17}, { 6, {  18, 5},  15}, { 7, {   9, 4},  26}},
/*   43*/{{ 2, {   5, 1},   5}, { 3, {  33, 2},  23}, { 4, {  22, 1},  29},
            { 5, {  27, 3},  13}, { 6, {  32, 1},  30}, { 7, {  30, 2},  38}},
/*   47*/{{ 2, {  31, 1},  44}, { 3, {  42, 1},  32}, { 4, {  10, 1},  45},
            { 5, {   5, 4},  17}, { 6, {  38, 5},  41}, { 7, {  14, 6},  12}},
/*   53*/{{ 2, {  26, 1},  18}, { 3, {  38, 2},  22}, { 4, {  15, 1},  27},
            { 5, {  48, 1},  21}, { 6, {  31, 5},  32}, { 7, {  18, 2},   5}},
/*   59*/{{ 2, {  14, 1},  50}, { 3, {  27, 1},  51}, { 4, {  58, 1},  14},
            { 5, {  39, 1},  27}, { 6, {  37, 1},  55}, { 7, {  22, 2},  26}},
/*   61*/{{ 2, {  44, 1},  26}, { 3, {  12, 1},   6}, { 4, {  39, 1},  55},
            { 5, {  19, 4},  18}, { 6, {  23, 5},   7}, { 7, {  32, 6},  55}},
/*   67*/{{ 2, {  60, 1},  18}, { 3, {  21, 2},  36}, { 4, {  39, 1},  28},
            { 5, {  53, 2},  16}, { 7, {   9, 1},  39},                 {0,{0,0},0}},
/*   71*/{{ 2, {  35, 1},  33}, { 3, {  63, 2},  29}, { 4, {  67, 3},  53},
            { 5, {  13, 1},  50}, { 6, {  69, 1},  31}, { 7, {  26, 4},  58}},
/*   73*/{{ 2, {  40, 1},  28}, { 3, {  58, 1},  29}, { 4, {   1, 1},  20},
            { 5, {   9, 3},  40}, { 6, {  19, 5},  39}, { 7, {  50, 2},  39}},
/*   79*/{{ 2, {   2, 1},  47}, { 3, {  49, 2},  76}, { 4, {  67, 3},  34},
            { 5, {  78, 1},  42}, { 6, {  50, 5},  53}, { 7, {  32, 4},   5}},
/*   83*/{{ 2, {   1, 1},  19}, { 3, {  34, 1},  33}, { 4, {  37, 1},  15},
            { 5, {  51, 1},  63}, { 6, {  30, 5},   5}, { 7, {  17, 2},  64}},
/*   89*/{{ 2, {  43, 1},  30}, { 3, {  29, 1},  24}, { 4, {  55, 3},  19},
            { 5, {  65, 2},  35}, { 6, {  75, 1},  83}, { 7, {  67, 6},  61}},
/*   97*/{{ 2, {  74, 1},  80}, { 3, {  52, 2},  38}, { 4, {  23, 3},   5},
            { 5, {  83, 1},  76}, { 6, {  11, 5},  84}, { 7, {  71, 1},  83}},
/*  101*/{{ 2, {  45, 1},  59}, { 3, {  30, 2},  63}, { 4, {  98, 1},  75},
            { 5, {  63, 1},  55}, { 6, {  63, 5},  90}, { 7, {  53, 3},  26}},
/*  103*/{{ 2, {  57, 1},  48}, { 3, {  19, 2},  49}, { 4, {  61, 1},  85},
            { 5, {  22, 1},  55}, { 6, {  96, 1},  67}, { 7, {  17, 3},  83}},
/*  107*/{{ 2, {  78, 1},  98}, { 3, {  56, 2},  29}, { 4, {  39, 1},  78},
            { 5, {  36, 3},  52}, { 6, {   7, 5},  73}, { 7, {  83, 1},  64}},
/*  109*/{{ 2, {  24, 1},  18}, { 3, {  52, 2},  95}, { 4, {  57, 3},  58},
            { 5, {  45, 1},  95}, { 6, {  62, 5},  95}, { 7, {  44, 5},  47}},
/*  113*/{{ 2, {  36, 1},  33}, { 3, {  11, 2},  59}, { 4, {  94, 3},  45},
            { 5, {  82, 1},  80}, { 6, {  84, 5},  12}, { 7, {  22, 2},  21}},
/*  127*/{{ 2, {  13, 1}, 116}, { 3, { 125, 2},  13}, { 4, {  42, 1},  56},
            { 5, {  99, 3}, 113}, { 6, {  86, 5},  48}, { 7, {  40, 3},  62}},
/*  131*/{{ 2, {  47, 1},  72}, { 3, {   5, 2},  25}, { 4, {  48, 1},  23},
            { 5, {  99, 2}, 100}, { 6, {  33, 5},   2}, { 7, { 115, 2},  81}},
/*  137*/{{ 2, {  72, 1},  21}, { 3, { 128, 1},  43}, { 4, {  67, 3},  57},
            { 5, {  80, 4}, 108}, { 6, {  22, 5},  33}, { 7, {  31, 4},  92}},
/*  139*/{{ 2, { 105, 1},  88}, { 3, { 126, 1},  37}, { 4, {  24, 1},  50},
            { 5, {  50, 3},  35}, { 6, {   8, 1}, 104}, { 7, { 123, 5}, 122}},
/*  149*/{{ 2, {  13, 1},  43}, { 3, {  22, 1},   2}, { 4, { 118, 1},  66},
            { 5, {   8, 2},  77}, { 6, {  74, 5}, 111}, { 7, {  20, 3}, 126}},
/*  151*/{{ 2, {  74, 1},   7}, { 3, {  64, 1},  40}, { 4, {   2, 1}, 146},
            { 5, { 122, 1},  18}, { 6, {  91, 1},  89}, { 7, {  23, 5},  36}},
/*  157*/{{ 2, {  63, 1},  96}, { 3, {  38, 1},  55}, { 4, { 126, 3},  55},
            { 5, { 131, 2},  38}, { 6, { 145, 1},  66}, { 7, { 137, 5}, 133}},
/*  163*/{{ 2, {  87, 1},  92}, { 3, { 111, 2},  62}, { 4, {  81, 3}, 159},
            { 5, {   1, 3},  33}, { 6, {  67, 5},  45}, { 7, {  87, 6},  35}},
/*  167*/{{ 2, {  33, 1},  45}, { 3, { 113, 1},   7}, { 4, { 109, 1},  40},
            { 5, {  33, 1}, 124}, { 6, {   1, 5}, 155}, { 7, {  85, 1},  42}},
```

```
/*  173*/{{ 2, {  62, 1},  70}, { 3, { 171, 1},  70}, { 4, {  17, 1},  71},
            { 5, { 143, 4},  97}, { 6, { 126, 5},  19},          {0,{0,0},0}},
/*  179*/{{ 2, {   5, 1},  98}, { 3, {  90, 1},  15}, { 4, {  44, 1}, 162},
            { 5, {  16, 2}, 173}, { 6, {  96, 1}, 175}, { 7, { 156, 3}, 110}},
/*  181*/{{ 2, { 168, 1},  47}, { 3, { 176, 1},  76}, { 4, { 100, 3}, 158},
            { 5, { 171, 4},  23}, { 6, {  11, 1},  97}, { 7, { 143, 3},  85}},
/*  191*/{{ 2, { 131, 1},  62}, { 3, {   9, 1}, 120}, { 4, {  35, 1}, 168},
            { 5, {  34, 3},  59}, { 6, {  78, 1}, 143}, { 7, { 166, 6},  79}},
/*  193*/{{ 2, { 170, 1}, 152}, { 3, { 170, 1}, 103}, { 4, { 124, 1},  22},
            { 5, {  20, 3}, 116}, { 6, {  38, 5},  58}, { 7, {  36, 5},  41}},
/*  197*/{{ 2, { 183, 1}, 162}, { 3, { 147, 1},  95}, { 4, { 111, 3},  67},
            { 5, {  39, 3},  32}, { 6, {  66, 1}, 166}, { 7, { 144, 2},  45}},
/*  199*/{{ 2, { 183, 1}, 190}, { 3, { 130, 1}, 115}, { 4, {  96, 1}, 108},
            { 5, { 129, 1},  50}, { 6, {  49, 1}, 192}, { 7, { 198, 4}, 128}},
/*  211*/{{ 2, { 197, 1},  85}, { 3, { 140, 1},  44}, { 4, {  41, 1}, 141},
            { 5, { 139, 4},  45}, { 6, {  88, 1},  57}, { 7, {  58, 5},  81}},
/*  223*/{{ 2, {  34, 1},  67}, { 3, {  52, 1}, 177}, { 4, {   1, 1}, 173},
            { 5, {  19, 2}, 218}, { 6, { 105, 1},  45}, { 7, { 181, 6}, 211}},
/*  227*/{{ 2, {  68, 1},  98}, { 3, {  53, 1},  97}, { 4, { 138, 3}, 115},
            { 5, {  93, 2}, 121}, { 6, { 220, 1}, 168}, { 7, { 188, 3}, 175}},
/*  229*/{{ 2, { 184, 1},  31}, { 3, {   6, 2}, 191}, { 4, {  59, 3}, 179},
            { 5, { 226, 2}, 152}, { 6, { 131, 5},  28}, { 7, { 122, 6},   6}},
/*  233*/{{ 2, {  17, 1}, 212}, { 3, {  86, 2}, 151}, { 4, {  30, 1},  43},
            { 5, { 219, 2}, 186}, { 6, { 172, 5}, 190}, { 7, {  60, 1},  20}},
/*  239*/{{ 2, { 233, 1}, 179}, { 3, {  33, 2},   4}, { 4, {  66, 1},  69},
            { 5, { 164, 3}, 113}, { 6, { 137, 5},  65}, { 7, {   7, 1}, 155}},
/*  241*/{{ 2, { 210, 1},  31}, { 3, { 178, 2}, 234}, { 4, {  73, 3}, 132},
            { 5, {  32, 4}, 142}, { 6, { 205, 5}, 112}, { 7, { 124, 2}, 199}},
/*  251*/{{ 2, { 204, 1}, 178}, { 3, { 133, 1}, 112}, { 4, {  28, 3}, 185},
            { 5, { 175, 3}, 190}, { 6, { 140, 1}, 220}, { 7, { 183, 5},  52}},
/*  257*/{{ 2, {   1, 1},   5}, { 3, {   1, 1},   5}, { 4, { 240, 3}, 218},
            { 5, {   1, 1},  10}, { 7, {  25, 5}, 132}, {13, {   1, 1},  19}},
/*  263*/{{ 2, { 110, 1}, 224}, { 3, {  49, 2},  74}, { 4, { 221, 3}, 194},
            { 5, { 228, 4},  39}, { 6, { 235, 5}, 215}, { 7, { 239, 6},  62}},
/*  269*/{{ 2, { 125, 1},  76}, { 3, { 223, 2}, 107}, { 4, { 194, 1}, 250},
            { 5, { 162, 1},  75}, { 6, { 169, 1}, 259}, { 7, { 100, 2}, 109}},
/*  271*/{{ 2, { 150, 1}, 182}, { 3, {  75, 1}, 180}, { 4, {   3, 3},  95},
            { 5, { 127, 4}, 112}, { 6, {  74, 5}, 161},          {0,{0,0},0}},
/*  277*/{{ 2, { 139, 1}, 178}, { 3, {  75, 1}, 259}, { 4, { 139, 1}, 205},
            { 5, { 242, 2}, 221}, { 6, { 175, 1},  43}, { 7, { 218, 5}, 197}},
/*  281*/{{ 2, {  83, 1},  26}, { 3, { 201, 1},  22}, { 4, { 141, 3},  23},
            { 5, { 200, 4}, 278}, { 6, { 117, 1}, 227}, { 7, { 248, 6}, 174}},
/*  283*/{{ 2, { 181, 1}, 153}, { 3, { 261, 2},  99}, { 4, {  28, 3}, 220},
            { 5, { 123, 4}, 117}, { 6, { 277, 5}, 249}, { 7, { 137, 4}, 129}},
/*  293*/{{ 2, {  43, 1}, 101}, { 3, {  36, 1},  98}, { 4, { 215, 3}, 219},
            { 5, { 219, 1}, 208}, { 6, { 241, 1}, 168},          {0,{0,0},0}},
/*  307*/{{ 2, { 288, 1}, 186}, { 3, { 224, 1},  87}, { 4, {  38, 1}, 161},
            { 5, {  55, 3}, 177}, { 6, { 124, 1}, 106}, { 7, { 154, 3},  39}},
/*  311*/{{ 2, { 279, 1}, 205}, { 3, { 227, 2}, 267}, { 4, { 144, 3}, 230},
            { 5, { 155, 3}, 160}, { 6, { 305, 1},  22}, { 7, { 130, 1}, 235}},
/*  313*/{{ 2, { 187, 1},  45}, { 3, {  64, 1},  31}, { 4, {  71, 1}, 167},
            { 5, { 233, 4}, 110}, { 6, { 256, 5},  34}, { 7, {  50, 1}, 130}},
/* 3187*/{{ 2, {   1, 1},   5}, { 3, {   1, 1},  25}, { 4, {   1, 1},   5},
            { 5, {   1, 1},  44}, { 6, {   1, 1},  23}, { 7, {   1, 1},  25}},
/*65537*/{{ 2, {   1, 1},   3}, { 3, {   1, 1},   7}, { 5, {   1, 1},   3},
            { 7, {   1, 1},   3}, {13, {   1, 1},  56}, {15, {   1, 1},  20}}
};

polynomial PrimitivePolynomialsTable [POLYNOMIALS_TABLE_SIZE] =
{
  {    2, 2,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 1},/* 1*/
  {    2, 8,{{ 1,6},{ 1,5},{ 1,3},          {0,0},{0,0},{0,0},{0,0}}, 1},/* 2*/
  {    2,15,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 1},/* 3*/
  {    2,59,{{ 1,6},{ 1,5},{ 1,4},{ 1,3},{1,1},          {0,0},{0,0}}, 1},/* 4*/
  {    2,61,{{ 1,5},{ 1,2},{ 1,1},          {0,0},{0,0},{0,0},{0,0}}, 1},/* 5*/
  {    2,67,{{ 1,5},{ 1,2},{ 1,1},          {0,0},{0,0},{0,0},{0,0}}, 1},/* 6*/
  {    3, 2,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 2},/* 7*/
  {    3, 3,{{ 2,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 1},/* 8*/
  {    5, 4,{{ 4,2},{ 4,1},          {0,0},{0,0},{0,0},{0,0},{0,0}}, 3},/* 9*/
  {    5, 8,{{ 3,7},{ 4,5},{ 1,4},{ 2,3},{ 4,2},{ 1,1},          {0,0}}, 3},/*10*/
  {    5,15,{{ 1,2},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 2},/*11*/
  {    5,19,{{ 1,3},{ 2,1},          {0,0},{0,0},{0,0},{0,0},{0,0}}, 2},/*12*/
  {    5,59,{{ 1,3},{ 2,2},{ 1,1},          {0,0},{0,0},{0,0},{0,0}}, 3},/*13*/
  {    5,61,{{ 1,2},{ 2,1},          {0,0},{0,0},{0,0},{0,0},{0,0}}, 3},/*14*/
  {    5,67,{{ 1,4},{ 1,3},{ 1,1},          {0,0},{0,0},{0,0},{0,0}}, 3},/*15*/
  {    7, 4,{{ 3,3},{ 5,2},{ 2,1},          {0,0},{0,0},{0,0},{0,0}}, 5},/*16*/
  {    7, 6,{{ 1,5},{ 2,3},{ 3,2},{ 5,1},          {0,0},{0,0},{0,0}}, 5},/*17*/
  {   11, 6,{{10,1},{ 5,2},{ 1,3},{ 8,4},{ 9,5},          {0,0},{0,0}}, 7},/*18*/
  {   11, 8,{{ 4,7},{ 6,6},{ 2,5},{ 1,4},{ 7,3},{ 1,2},{ 1,1}     }, 8},/*19*/
```

```
234   {   13, 4,{{ 5,3},{ 7,2},{ 6,1},         {0,0},{0,0},{0,0},{0,0}}, 6},/*20*/
235   {   13, 6,{{ 2,5},{ 9,3},{ 5,2},{10,1},        {0,0},{0,0},{0,0}}, 6},/*21*/
236   {   13, 8,{{ 3,7},{ 3,6},{ 5,5},{10,4},{ 6,3},{10,2},     {0, 0}}, 2},/*22*/
237   {   17,13,{{ 2,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 6},/*23*/
238   {   17,15,{{ 3,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 6},/*24*/
239   {   17,19,{{ 3,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 6},/*25*/
240   {   17,58,{{ 3,2},{ 3,1},          {0,0},{0,0},{0,0},{0,0},{0,0}},11},/*26*/
241   {   17,61,{{ 2,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 6},/*27*/
242   {   17,67,{{ 1,2},{11,1},          {0,0},{0,0},{0,0},{0,0},{0,0}}, 3},/*28*/
243   {   67, 6,{{32,5},{ 3,4},{30,3},{39,2},{10,1},       {0,0},{0,0}},48},/*29*/
244   {  257,15,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}},43},/*30*/
245   {  257,19,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}, 7},/*31*/
246   {  257,59,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}},85},/*32*/
247   {65537,19,{{ 1,1},           {0,0},{0,0},{0,0},{0,0},{0,0},{0,0}},47} /*33*/
248 };
249
250 #endif  /* _PRIMITIVE_TABLES_H */
```

Universitat d'Alacant
Universidad de Alicante

B.2 Generador con caja de sustitución

Las instrucciones para el correcto uso y paso de parámetros, que se muestran a continuación, se obtienen al invocar el ejecutable sin parámetros:

```
Usage: Msbox l r s t i [file [-q | -t]]

        l:      Sequence bits length (0 = endless)
                It can be followed by B, KB, MB, GB, TB or PB for
                bytes, kilobytes, megabytes, gigabytes or petabytes
        r s t:  A B C matrices sizes
        i:      Random key identification (call number)
        file:   Tracker output file name (optional)
        -q:     Quiet tracker (Only arguments and end-begin time)
        -t:     Quiet tracker and no output sequence (Only to test time)
```

Esta implementación del generador usa el mismo fichero de cabecera para las tablas de polinomios primitivos que en la versión anterior. Salvo ese fichero, que puede consultarse en el apéndice B.1, el resto del código se muestra a continuación, aunque solo con la intención de que se use para alguna aclaración o consulta rápida. Para la compilación del código o su reutilización se facilita en el siguiente enlace todos los ficheros necesarios, incluido también el fichero de cabecera con las tablas de polinomios primitivos mostrado en la sección anterior:

http://www.dccia.ua.es/~fmartine/tesis/Msbox.zip

El enlace incluye también un proyecto para el entorno Dev-C++ que facilita la compilación conjunta del módulo motor del generador con el programa.

Universidad de Alicante

Msbox.c:

```
/*
  MSBOX: Pseudorandom number generator based upon 3x3 block upper
    triangular matrix with 32x32 s-boxes filtration and p=65537
    (Matrices TSB de 3x3 bloques con filtro s-box 32x32 y p=65537).

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/


/* headers */

#include <stdlib.h>
#include <stdint.h>
#include <inttypes.h>
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>


#include "motor.h"


/* arguments */
typedef struct
{
  uint64_t l;
  uint_fast8_t r;
  uint_fast8_t s;
  uint_fast8_t t;
  uint_fast8_t i;
  char * fileName;
  uint_fast8_t quiet;
  uint_fast8_t test;
} arguments;

arguments * GetArguments(int argc, char * argv[]);
void DropArguments(arguments * args);
void Usage(char * command);
void PrintArguments(FILE * file, arguments * args);


/* others prototypes */

FILE * OutputFile();
FILE * OpenTrackRegister(arguments * args);
void CloseTrackRegister(FILE * file);
void TimeStamp(FILE * file, char * message);

void GenerateBits(FILE * f, motor m, uint64_t bits);


/* code */

int main(int argc, char * argv[])
{
  arguments * args;
  FILE * file;
  key k;
  LARGE_INTEGER li;
  uint64_t r, f;
  motor m;


  args = GetArguments(argc, argv);
  if (args->fileName != NULL)
    file = OpenTrackRegister(args);
  else
    file = NULL;

  k = CreateKey(args->i);
  if (!args->quiet)
    PrintKey(file, k);
  else
```

Universitat d'Alacant
Universidad de Alicante

```
77    {
78      QueryPerformanceCounter(&li);
79      r = (uint64_t)li.QuadPart;
80    }
81
82    m = CreateMotor(args->r, args->s, args->t, k, (args->quiet)?NULL:file);
83    GenerateBits((args->test)?NULL:OutputFile(), m, args->l);
84
85    DeleteMotor(m);
86    DeleteKey(k);
87
88    if (file)
89    {
90      if (args->quiet)
91      {
92        QueryPerformanceCounter(&li);
93        r = (uint64_t)li.QuadPart - r;
94        QueryPerformanceFrequency(&li);
95        f = li.QuadPart;
96        fprintf(file,"End-Begin Time:\n\tCycles:    %15"PRIu64"\n", r);
97        fprintf(file,"\tFrequency: %15"PRIu64"\n\tSeconds:  %16.8lf\n",
98                f, (double)r/f);
99      }
100     CloseTrackRegister(file);
101   }
102   DropArguments(args);
103
104   return EXIT_SUCCESS;
105 }
106
107
108 void GenerateBits(FILE * f, motor m, uint64_t bits)
109 {
110   uint64_t n;
111   uint8_t * bytes;
112   uint_fast16_t size;
113
114   size = SizeOfExtraction(m);
115   bytes = (uint8_t *)malloc(size);
116   n = (bits) ? (bits>>3) : -1;
117   while (n)
118   {
119     GetExtraction(m,bytes);
120     if (n<size) size = n;
121     if (f != NULL)
122       if (!fwrite(bytes,size,1,f))
123       {
124         fprintf(stderr,"Error writing the output file.\n");
125         exit(EXIT_FAILURE);
126       }
127     n-=size;
128   }
129   free(bytes);
130 }
131
132
133 FILE * OutputFile()
134 {
135   if(-1 == _setmode(_fileno(stdout),_O_BINARY))
136     fprintf(stderr,"Cannot set stdout binary mode.\n");
137   return stdout;
138 }
139
140
141 FILE * OpenTrackRegister(arguments * args)
142 {
143   FILE * file;
144
145   file = fopen(args->fileName, "w");
146   if (!file)
147   {
148     fprintf(stderr, "Error opening the tracker file.\n");
149     exit(EXIT_FAILURE);
150   }
151   PrintArguments(file, args);
152   TimeStamp(file, "Begin");
153   return file;
154 }
155
```

Universitat d'Alacant
Universidad de Alicante

```
156
157 void CloseTrackRegister(FILE * file)
158 {
159   TimeStamp(file, "End");
160   fclose(file);
161 }
162
163 void TimeStamp(FILE * file, char * message)
164 {
165   char time_buff[9], date_buff[9];
166
167   if (_strtime_s(time_buff,9) || _strdate_s(date_buff,9))
168   {
169     fprintf(stderr, "Time/Date system error.\n");
170     exit(EXIT_FAILURE);
171   }
172   else
173     fprintf(file, "\n%s: %s %s\n\n", message, date_buff, time_buff);
174 }
175
176
177 arguments * GetArguments(int argc, char * argv[])
178 {
179   arguments * args;
180   char * end;
181   uint8_t i;
182   const uint64_t B = 8;
183   const uint64_t K = 1024 * B;
184   const uint64_t M = 1024 * K;
185   const uint64_t G = 1024 * M;
186   const uint64_t T = 1024 * G;
187   const uint64_t P = 1024 * T;
188
189   args = (arguments *)malloc(sizeof(arguments));
190   if ((argc < 6) || (argc > 8)) Usage(argv[0]);
191
192   i=0;
193   args->l = strtoumax(argv[++i],&end,10);
194   if (!strcmp("B",end)) args->l*=B; else
195     if (!strcmp("KB",end)) args->l*=K; else
196       if (!strcmp("MB",end)) args->l*=M; else
197         if (!strcmp("GB",end)) args->l*=G; else
198           if (!strcmp("TB",end)) args->l*=T; else
199             if (!strcmp("PB",end)) args->l*=P; else
200               if (strlen(end) > 0) Usage(argv[0]);
201   args->r = strtoul(argv[++i],&end,10);
202   if ((args->r == 0) || strlen(end) > 0) Usage(argv[0]);
203   args->s = strtoul(argv[++i],&end,10);
204   if ((args->s == 0) || strlen(end) > 0) Usage(argv[0]);
205   args->t = strtoul(argv[++i],&end,10);
206   if ((args->t == 0) || strlen(end) > 0) Usage(argv[0]);
207   args->i = strtoul(argv[++i],&end,10);
208   if (strlen(end) > 0) Usage(argv[0]);
209
210   if (argc > 6)
211   {
212     args->fileName = argv[++i];
213     if (argc == 8)
214     {
215       i++;
216       if (strcmp("-q",argv[i]) && strcmp("-t",argv[i])) Usage(argv[0]);
217       args->quiet = 1;
218       args->test = !strcmp("-t",argv[i]);
219     }
220     else
221     {
222       args->quiet = 0;
223       args->test = 0;
224     }
225   }
226   else
227   {
228     args->fileName = NULL;
229     args->quiet = 0;
230     args->test = 0;
231   }
232
233   return args;
234 }
```

```
235
236
237 void DropArguments(arguments * args)
238 {
239   free(args);
240 }
241
242
243 void Usage(char * command)
244 {
245   char drive[_MAX_DRIVE];
246   char dir[_MAX_DIR];
247   char fname[_MAX_FNAME];
248   char ext[_MAX_EXT];
249   errno_t err;
250
251   err = _splitpath_s(command,drive,_MAX_DRIVE,dir,_MAX_DIR,
252                      fname,_MAX_FNAME,ext,_MAX_EXT);
253   if (err != 0)
254   {
255     fprintf(stderr,"Error splitting the path. Error code %d.\n", err);
256     exit(EXIT_FAILURE);
257   }
258   fprintf(stderr,"\nUsage: %s l r s t i [file [-q | -t]]\n\n",fname);
259   fprintf(stderr,"\tl:\tSequence bits length (0 = endless)\n");
260   fprintf(stderr,"\t  \tIt can be followed by B, KB, MB, GB, TB or PB for\n");
261   fprintf(stderr,"\t  \tbytes, kilobytes, megabytes, gigabytes or petabytes\n");
262   fprintf(stderr,"\tr s t:\tA B C matrices sizes\n");
263   fprintf(stderr,"\ti:\tRandom key identification (call number)\n");
264   fprintf(stderr,"\tfile:\tTracker output file name (optional)\n");
265   fprintf(stderr,"\t-q:\tQuiet tracker (Only arguments and end-begin time)\n");
266   fprintf(stderr,
267           "\t-t:\tQuiet tracker and no output sequence (Only to test time)\n");
268   fprintf(stderr,"\n");
269   exit(EXIT_FAILURE);
270 }
271
272
273 void PrintArguments(FILE * file, arguments * args)
274 {
275   fprintf(file,"Arguments:\n");
276   if (args->l)
277     fprintf(file,"\tSequence of %"PRIu64" bits\n", args->l);
278   else
279     fprintf(file,"\tEndless sequence of bits\n");
280   fprintf(file,
281           "\tMatrices sizes: r=%"PRIuFAST8" s=%"PRIuFAST8" t=%"PRIuFAST8"\n",
282           args->r,args->s,args->t);
283   fprintf(file,"\tCall identification %"PRIuFAST8". ", args->i);
284   fprintf(file,"Tracker output file: %s%s\n\n",
285           args->fileName,(args->quiet)?" (quiet)":"");
286 }
```

motor.h:

```
/*
  MOTOR: Header of types and functions for pseudorandom
    number generator based upon 3x3 block upper triangular
    matrix with 32x32 s-boxes filtration and p=65537.
  Version: Parameterizable with r, s, t.

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/

#ifndef _MOTOR_H
#define _MOTOR_H

/* headers */

#include <stdint.h>
#include <stdio.h>


/* key */
typedef uint8_t * key;
#define KEYBYTESLENGTH 256

key CreateKey(uint_fast8_t id);
void DeleteKey(key k);
void PrintKey(FILE * file, key k);

/* motor */

#ifndef _STRUCT_MOTOR
#define _STRUCT_MOTOR
#endif

typedef _STRUCT_MOTOR * motor; /* Pointer a private structure */

motor CreateMotor(uint_fast8_t r, uint_fast8_t s,
                  uint_fast8_t t, key k, FILE * file);
void DeleteMotor(motor m);
void PrintMotor(FILE * file, motor m);
uint_fast16_t SizeOfExtraction(motor m);
void GetExtraction(motor m, uint8_t * bytes);

#endif  /* _MOTOR_H */
```

Universitat d'Alacant
Universidad de Alicante

motor.c:

```
/*
  MOTOR: Implementation module of functions for pseudorandom
    number generator based upon 3x3 block upper triangular
    matrix with 32x32 s-boxes filtration and p=65537.
  Version: Parameterizable with r, s, t.

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/

#define _MOTOR_H

/* headers */

#include <stdint.h>
#include <inttypes.h>
#include <stdio.h>
#include <windows.h>

#include "primitive-tables.h"



/* define section */

#define PRIME 65537



/* types */

/* elements array */

typedef uint64_t * elements;


/* data */
typedef struct
{
  uint_fast16_t length;
  uint_fast16_t size;
  elements e;
} struct_data;
typedef struct_data * data;

data CreateData(uint_fast16_t l);
data LinkData(uint_fast16_t l, elements e);
void CopyData(data dest, data src);
void DeleteData(data dat);


/* matrix */
typedef struct
{
  uint_fast8_t rows;
  uint_fast8_t cols;
  data d;
} struct_matrix;
typedef struct_matrix * matrix;

matrix CreateMatrix(uint_fast8_t rows, uint_fast8_t cols);
matrix LinkMatrix(uint_fast8_t rows, uint_fast8_t cols, elements e);
void DeleteMatrix(matrix mat);
void PrintMatrix(FILE * file, matrix mat);
void FillAssociatedMatrix(matrix mat, uint_fast32_t p);
void FillPrimitiveVector(elements vec, uint_fast32_t p, uint_fast8_t d);
void CopyMatrix(matrix dest, matrix src);
void AddMatrices(matrix dest, matrix src1, matrix src2);
void MultMatrices(matrix dest, matrix src1, matrix src2);
void ModMatrix(matrix mat);


/* tracker */
typedef struct
```

Universitat d'Alacant
Universidad de Alicante

```
76  {
77    FILE * file;
78    uint64_t freq;
79    uint64_t first;
80    uint64_t last;
81    uint64_t initial;
82    uint64_t n_e;
83    uint64_t min_e;
84    uint64_t max_e;
85    double mean_e;
86    uint64_t n_f;
87    uint64_t min_f;
88    uint64_t max_f;
89    double mean_f;
90  } struct_tracker;
91  typedef struct_tracker * tracker;
92
93  tracker CreateTracker(FILE *file);
94  void DeleteTracker(tracker tr);
95  void PrintTimeRegister(tracker tr);
96  void RegisterBeginExtraction(tracker tr);
97  void RegisterEndExtraction(tracker tr);
98
99
100 /* s-boxes */
101 typedef uint8_t * sbox8x8;
102 typedef uint32_t * sbox8x32;
103 #define SB_ELEMENTS 256
104
105 typedef sbox8x32 * sboxes;
106 #define SB_SBOXES 4
107
108
109 /* motor */
110 typedef struct
111 {
112   sboxes S;
113   uint32_t filter;
114   data seed;
115   matrix X;
116   matrix Y;
117   matrix Z;
118   matrix A;
119   matrix B;
120   matrix C;
121   uint64_t h;
122   data extraction;
123   matrix Xh;
124   matrix Yh;
125   matrix Zh;
126   matrix Ah;
127   matrix Bh;
128   matrix auxX1;
129   matrix auxX2;
130   matrix auxX3;
131   matrix auxX4;
132   matrix auxY1;
133   matrix auxY2;
134   matrix auxY3;
135   matrix auxZ1;
136   matrix auxZ2;
137   matrix auxA;
138   matrix auxB;
139   tracker tr;
140 } struct_motor;
141
142 #define _STRUCT_MOTOR struct_motor
143 #undef _MOTOR_H
144 #include "motor.h"
145
146 void Iterate(motor m);
147 sboxes CreateSboxes(key k);
148 void DeleteSboxes(sboxes S);
149
150
151 /* seed (pseudo data) */
152 data CreateSeed(uint_fast16_t r, uint_fast16_t s, uint_fast16_t t);
153 void FillSeed(data seed, sboxes S, key k, uint_fast32_t prime);
154 void DeleteSeed(data seed);
```

Universitat d'Alacant
Universidad de Alicante

```
155
156
157
158   /* Code */
159
160
161   key CreateKey(uint_fast8_t id)
162   {
163     key k;
164     uint_fast16_t i;
165
166     k = (key)malloc(sizeof(uint8_t)*KEYBYTESLENGTH);
167
168     srand(101*id);
169     for (i=0; i < KEYBYTESLENGTH; i++) k[i] = rand() % 256;
170
171     return k;
172   }
173
174
175   void DeleteKey(key k)
176   {
177     free(k);
178   }
179
180
181   void PrintKey(FILE * file, key k)
182   {
183     uint_fast16_t i;
184
185     fprintf(file, "Key:");
186     for (i=0; i < KEYBYTESLENGTH; i++)
187         fprintf(file, "%s %03"PRIx8"", (i%20)?"":"\n", k[i]);
188     fprintf(file, "\n");
189   }
190
191
192   motor CreateMotor(uint_fast8_t r, uint_fast8_t s,
193                     uint_fast8_t t, key k, FILE * file)
194   {
195     motor m;
196     uint8_t i;
197     uint_fast16_t pos;
198
199     m = (motor)malloc(sizeof(struct_motor));
200
201     if (file != NULL)
202       m->tr = CreateTracker(file);
203     else
204       m->tr = NULL;
205
206     m->S = CreateSboxes(k);
207     m->filter = 0;
208
209     m->seed = CreateSeed(r,s,t);
210     FillSeed(m->seed, m->S, k, PRIME);
211
212     m->Y = LinkMatrix(r, s, m->seed->e);
213     pos = (r*s);
214     m->X = LinkMatrix(r, t, m->seed->e + pos);
215     pos += (r*t);
216     m->Z = LinkMatrix(s, t, m->seed->e + pos);
217
218
219     m->A = CreateMatrix(r, r);
220     m->B = CreateMatrix(s, s);
221     m->C = CreateMatrix(t, t);
222     FillAssociatedMatrix(m->A, PRIME);
223     FillAssociatedMatrix(m->B, PRIME);
224     FillAssociatedMatrix(m->C, PRIME);
225
226     m->extraction = CreateData(m->seed->length);
227     m->Yh = LinkMatrix(r, s, m->extraction->e);
228     pos = (r*s);
229     m->Xh = LinkMatrix(r, t, m->extraction->e + pos);
230     pos += (r*t);
231     m->Zh = LinkMatrix(s, t, m->extraction->e + pos);
232     m->Ah = CreateMatrix(r, r);
233     m->Bh = CreateMatrix(s, s);
```

Universitat d'Alacant
Universidad de Alicante

```
234
235       m->h = 1;
236       CopyMatrix(m->Xh, m->X);
237       CopyMatrix(m->Yh, m->Y);
238       CopyMatrix(m->Zh, m->Z);
239       CopyMatrix(m->Ah, m->A);
240       CopyMatrix(m->Bh, m->B);
241
242       m->auxX1 = CreateMatrix(r, t);
243       m->auxX2 = CreateMatrix(r, t);
244       m->auxX3 = CreateMatrix(r, t);
245       m->auxX4 = CreateMatrix(r, t);
246       m->auxY1 = CreateMatrix(r, s);
247       m->auxY2 = CreateMatrix(r, s);
248       m->auxY3 = CreateMatrix(r, s);
249       m->auxZ1 = CreateMatrix(s, t);
250       m->auxZ2 = CreateMatrix(s, t);
251       m->auxA = CreateMatrix(r, r);
252       m->auxB = CreateMatrix(s, s);
253
254       for (i=0; i < 64; i++) Iterate(m);
255
256       return m;
257     }
258
259
260     void DeleteMotor(motor m)
261     {
262       if (m->tr != NULL)
263       {
264         PrintMotor(m->tr->file, m);
265         PrintTimeRegister(m->tr);
266         DeleteTracker(m->tr);
267       }
268       DeleteSboxes(m->S);
269       DeleteData(m->seed);
270       DeleteMatrix(m->A);
271       DeleteMatrix(m->B);
272       DeleteMatrix(m->C);
273       DeleteData(m->extraction);
274       DeleteMatrix(m->Ah);
275       DeleteMatrix(m->Bh);
276       DeleteMatrix(m->auxX1);
277       DeleteMatrix(m->auxX2);
278       DeleteMatrix(m->auxX3);
279       DeleteMatrix(m->auxX4);
280       DeleteMatrix(m->auxY1);
281       DeleteMatrix(m->auxY2);
282       DeleteMatrix(m->auxY3);
283       DeleteMatrix(m->auxZ1);
284       DeleteMatrix(m->auxZ2);
285       DeleteMatrix(m->auxA);
286       DeleteMatrix(m->auxB);
287       free(m);
288     }
289
290
291     void PrintMotor(FILE * file, motor m)
292     {
293       fprintf(file, "M:\n");
294       fprintf(file, "X=\n"); PrintMatrix(file, m->X);
295       fprintf(file, "Y=\n"); PrintMatrix(file, m->Y);
296       fprintf(file, "Z=\n"); PrintMatrix(file, m->Z);
297       fprintf(file, "A=\n"); PrintMatrix(file, m->A);
298       fprintf(file, "B=\n"); PrintMatrix(file, m->B);
299       fprintf(file, "C=\n"); PrintMatrix(file, m->C);
300       fprintf(file, "M^h: h=%"PRIu64"\n", m->h);
301       fprintf(file, "X^h=\n"); PrintMatrix(file, m->Xh);
302       fprintf(file, "Y^h=\n"); PrintMatrix(file, m->Yh);
303       fprintf(file, "Z^h=\n"); PrintMatrix(file, m->Zh);
304       fprintf(file, "A^h=\n"); PrintMatrix(file, m->Ah);
305       fprintf(file, "B^h=\n"); PrintMatrix(file, m->Bh);
306     }
307
308
309     uint_fast16_t SizeOfExtraction(motor m)
310     {
311       return m->extraction->length * 4;
312     }
```

Universitat d'Alacant
Universidad de Alicante

```
313
314
315   void GetExtraction(motor m, uint8_t * bytes)
316   {
317     uint_fast16_t i, pos;
318
319     i=0;         /* Two iterations a extractrion */
320     if (m->tr != NULL)
321       RegisterBeginExtraction(m->tr);
322
323     Iterate(m);  /* First iteration */
324     for (pos=0; pos < m->extraction->length; pos++)
325     {
326       bytes[i++] = (m->extraction->e[pos])>>8;
327       bytes[i++] = m->extraction->e[pos];
328     }
329
330     Iterate(m);  /* Second iteration */
331     for (pos=0; pos < m->extraction->length; pos++)
332     {
333       bytes[i++] = (m->extraction->e[pos])>>8;
334       bytes[i++] = m->extraction->e[pos];
335     }
336
337     if (m->tr != NULL)
338       RegisterEndExtraction(m->tr);
339
340     i=0;         /* Filtration with s-box */
341     for (pos=0; pos < m->extraction->length; pos++)
342     {
343       m->filter ^= m->S[0][bytes[i++]];
344       m->filter ^= m->S[1][bytes[i++]];
345       m->filter ^= m->S[2][bytes[i++]];
346       m->filter ^= m->S[3][bytes[i]];
347       i-=3;
348       bytes[i++] = m->filter;
349       bytes[i++] = (m->filter)>>8;
350       bytes[i++] = (m->filter)>>16;
351       bytes[i++] = (m->filter)>>24;
352     }
353   }
354
355
356   void Iterate(motor m)
357   {
358     /* Almacenamiento provisional de Yh-1 en auxY3 */
359     CopyMatrix(m->auxY3, m->Yh);
360
361     /* Calculo de:  Yh = (Ah-1 · Y) + (Yh-1 · B) */
362     /*  como        Yh =   auxY1   +   auxY2     */
363     MultMatrices(m->auxY1, m->Ah, m->Y);
364     MultMatrices(m->auxY2, m->Yh, m->B);
365     AddMatrices(m->Yh, m->auxY1, m->auxY2);
366     ModMatrix(m->Yh);
367
368     /* Calculo de:  Xh = (Ah-1 · X) + (Yh-1 · Z) + (Xh-1 · C)  */
369     /*  como        Xh = (Ah-1 · X) + (auxY3 · Z) + (Xh-1 · C) */
370     /*              Xh  =  ( auxX1   +    auxX2 )  +   auxX3   */
371     /*              Xh  =         auxX4        +     auxX3     */
372     MultMatrices(m->auxX1, m->Ah, m->X);
373     MultMatrices(m->auxX2, m->auxY3, m->Z);
374     MultMatrices(m->auxX3, m->Xh, m->C);
375     AddMatrices(m->auxX4, m->auxX1, m->auxX2);
376     AddMatrices(m->Xh, m->auxX4, m->auxX3);
377     ModMatrix(m->Xh);
378
379     /* Calculo de:  Zh = (Bh-1 · Z) + (Zh-1 · C) */
380     /*  como        Zh  =  auxZ1   +   auxZ2     */
381     MultMatrices(m->auxZ1, m->Bh, m->Z);
382     MultMatrices(m->auxZ2, m->Zh, m->C);
383     AddMatrices(m->Zh, m->auxZ1, m->auxZ2);
384     ModMatrix(m->Zh);
385
386     /* Calculo de:  Ah = (Ah-1 · A) */
387     /*  como        Ah  =   auxA    */
388     MultMatrices(m->auxA, m->Ah, m->A);
389     CopyMatrix(m->Ah, m->auxA);
390     ModMatrix(m->Ah);
391
```

Universitat d'Alacant
Universidad de Alicante

```
392       /* Calculo de:  Bh = (Bh-1 · B) */
393       /*  como        Bh  =   auxB    */
394       MultMatrices(m->auxB, m->Bh, m->B);
395       CopyMatrix(m->Bh, m->auxB);
396       ModMatrix(m->Bh);
397
398       m->h++;
399     }
400
401
402     sboxes CreateSboxes(key k)
403     {
404       sbox8x8 s;
405       sboxes S;
406       uint_fast16_t i, h;
407       uint_fast8_t j, c;
408       uint32_t t;
409
410       s = (sbox8x8)malloc(sizeof(uint8_t)*SB_ELEMENTS);
411       S = (sboxes)malloc(sizeof(sbox8x32)*SB_SBOXES);
412       for (j=0; j < SB_SBOXES; j++)
413         S[j] = (sbox8x32)malloc(sizeof(uint32_t)*SB_ELEMENTS);
414
415       for (i=0; i < SB_ELEMENTS; i++)
416       {
417         s[i] = i;
418         for (j=0; j < SB_SBOXES; j++)
419           S[j][i] = 0;
420       }
421
422       for (j=0; j < SB_SBOXES; j++)
423         for (c=0; c < 4; c++)
424         {
425           h=0;
426           for (i=0; i < SB_ELEMENTS; i++)
427           {
428             h = (h + s[i] + k[i%KEYBYTESLENGTH]) % SB_ELEMENTS;
429             t = s[i];
430             s[i] = s[h];
431             s[h] = t;
432           }
433           for (i=0; i < SB_ELEMENTS; i++)
434             S[j][i] = (S[j][i]<<8) ^ s[i];
435         }
436
437       free(s);
438       return S;
439     }
440
441
442     void DeleteSboxes(sboxes S)
443     {
444       free(S);
445     }
446
447
448     data CreateSeed(uint_fast16_t r, uint_fast16_t s, uint_fast16_t t)
449     {
450       return CreateData(r*t+r*s+s*t);
451     }
452
453
454     void FillSeed(data seed, sboxes S, key k, uint_fast32_t prime)
455     {
456       uint_fast16_t i, j;
457       uint64_t v;
458
459       j = 0;
460       seed->e[0] = 0x5A5A;
461       for (i=1; i < seed->length-1; i++)
462       {
463         v = S[0][(i+k[j%16])%256];
464         j++;
465         v ^= S[1][(j+k[j%16])%256];
466         j++;
467         v<<=32;
468         v ^= S[2][(i+k[j%16])%256];
469         j++;
470         v ^= S[3][(j+k[j%16])%256];
```

Universitat d'Alacant
Universidad de Alicante

```
471         j++;
472         seed->e[i] = v % prime;
473       }
474       seed->e[i] = 0xA5A5;
475     }
476
477
478     void DeleteSeed(data seed)
479     {
480       DeleteData(seed);
481     }
482
483
484     tracker CreateTracker(FILE * file)
485     {
486       struct_tracker * tr;
487       LARGE_INTEGER li;
488
489       tr = (tracker)malloc(sizeof(struct_tracker));
490       tr->file = file;
491       QueryPerformanceFrequency(&li);
492       tr->freq = li.QuadPart;
493       QueryPerformanceCounter(&li);
494       tr->first = li.QuadPart;
495       tr->last = li.QuadPart;
496       tr->initial = 0;
497       tr->n_e = 0;
498       tr->min_e = UINT64_MAX;
499       tr->max_e = 0;
500       tr->mean_e = 0.0;
501       tr->n_f = 0;
502       tr->min_f = UINT64_MAX;
503       tr->max_f = 0;
504       tr->mean_f = 0.0;
505       return tr;
506     }
507
508
509     void DeleteTracker(tracker tr)
510     {
511       free(tr);
512     }
513
514
515     void PrintTimeRegister(tracker tr)
516     {
517       RegisterBeginExtraction(tr); /* End of no extraccion actions (filtrations) */
518       fprintf(tr->file,"Performance:      Frequency: %"PRIu64"\n", tr->freq);
519       fprintf(tr->file,"Initialitation:  Time cycles: %"PRIu64" Seconds: %.8lf\n",
520               tr->initial,(double)tr->initial/tr->freq);
521       fprintf(tr->file,"Extractions: %"PRIu64"\n", tr->n_e);
522       fprintf(tr->file,"             Min. Time cycles: %"PRIu64" Seconds: %.8lf\n",
523               tr->min_e, (double)tr->min_e/tr->freq);
524       fprintf(tr->file,"             Max. Time cycles: %"PRIu64" Seconds: %.8lf\n",
525               tr->max_e, (double)tr->max_e/tr->freq);
526       fprintf(tr->file,"             Mean Time cycles: %.0lf Seconds: %.8lf\n",
527               tr->mean_e, tr->mean_e/tr->freq);
528       fprintf(tr->file,"Filtrations: %llu\n", tr->n_f);
529       fprintf(tr->file,"             Min. Time cycles: %"PRIu64" Seconds: %.8lf\n",
530               tr->min_f, (double)tr->min_f/tr->freq);
531       fprintf(tr->file,"             Max. Time cycles: %"PRIu64" Seconds: %.8lf\n",
532               tr->max_f, (double)tr->max_f/tr->freq);
533       fprintf(tr->file,"             Mean Time cycles: %.0lf Seconds: %.8lf\n",
534               tr->mean_f, tr->mean_f/tr->freq);
535       fprintf(tr->file,"        Counter:       First: %15"PRIu64"\n", tr->first);
536       fprintf(tr->file,"                        Last: %15"PRIu64"\n", tr->last);
537       fprintf(tr->file,
538               "        Total:   Time cycles: %15"PRIu64" Seconds: %.8lf\n\n",
539               tr->last-tr->first,(double)(tr->last-tr->first)/tr->freq);
540     }
541
542
543     void RegisterBeginExtraction(tracker tr)
544     {
545       LARGE_INTEGER li;
546       uint64_t r;
547
548       QueryPerformanceCounter(&li);
549       if (tr->initial)
```

```
550       {
551         r = (uint64_t)li.QuadPart-tr->last;
552         tr->last+=r;
553         if (r < tr->min_f) tr->min_f=r;
554         if (r > tr->max_f) tr->max_f=r;
555         tr->mean_f=((tr->mean_f*tr->n_f)+r)/++(tr->n_f);
556       }
557       else
558       {
559         r = (uint64_t)li.QuadPart;
560         tr->initial=r-tr->last;
561         tr->last=r;
562       }
563     }
564
565
566     void RegisterEndExtraction(tracker tr)
567     {
568       LARGE_INTEGER li;
569       uint64_t r;
570
571       QueryPerformanceCounter(&li);
572       r = (uint64_t)li.QuadPart-tr->last;
573       tr->last+=r;
574       if (r < tr->min_e) tr->min_e=r;
575       if (r > tr->max_e) tr->max_e=r;
576       tr->mean_e=((tr->mean_e*tr->n_e)+r)/++(tr->n_e);
577     }
578
579
580     matrix CreateMatrix(uint_fast8_t rows, uint_fast8_t cols)
581     {
582       matrix mat;
583
584       mat = (matrix)malloc(sizeof(struct_matrix));
585       mat->rows = rows;
586       mat->cols = cols;
587       mat->d = CreateData(rows*cols);
588       return mat;
589     }
590
591
592     matrix LinkMatrix(uint_fast8_t rows, uint_fast8_t cols, elements e)
593     {
594       matrix mat;
595
596       mat = (matrix)malloc(sizeof(struct_matrix));
597       mat->rows = rows;
598       mat->cols = cols;
599       mat->d = LinkData(rows*cols, e);
600       return mat;
601     }
602
603
604     void DeleteMatrix(matrix mat)
605     {
606       DeleteData(mat->d);
607       free(mat);
608     }
609
610
611     void PrintMatrix(FILE * file, matrix mat)
612     {
613       uint_fast8_t r, c;
614       uint_fast16_t pos;
615
616       pos = 0;
617       fprintf(file,"\n");
618       for(r=0; r<mat->rows; r++)
619       {
620         fprintf(file,"[ ");
621         for(c=0; c<mat->cols; c++)
622           fprintf(file,"%6"PRIu64"",mat->d->e[pos++]);
623         fprintf(file,"]\n");
624       }
625       fprintf(file,"\n");
626     }
627
628
```

Universitat d'Alacant
Universidad de Alicante

```
629  void FillAssociatedMatrix(matrix mat, uint_fast32_t p)
630  {
631    uint_fast8_t r, c;
632    uint_fast16_t pos;
633
634    pos = 0;
635    for (r=0; r<mat->rows; r++)
636      for (c=0; c<mat->cols; c++)
637        if (c==r+1)
638          mat->d->e[pos++]=1;
639        else
640          mat->d->e[pos++]=0;
641    FillPrimitiveVector(&(mat->d->e[(r-1)*c]),p,c);
642  }
643
644
645  void FillPrimitiveVector(elements vec, uint_fast32_t p, uint_fast8_t d)
646  {
647    unsigned i, j;
648
649    for (i=0; (i<PRIMES_TABLE_SIZE) && (PrimesTable[i]<p); i++);
650    if ((i==PRIMES_TABLE_SIZE) || (PrimesTable[i]>p))
651    {
652      fprintf(stderr,"Error: Prime p=%"PRIuFAST32
653              " not found to get a primitive polynomial.\n",p);
654      exit(EXIT_FAILURE);
655    }
656    if (d==1)
657    {
658      vec[0]=PrimitiveRootsTable[i];
659      return;
660    }
661    for (j=0;
662         (j<MAX_TRINOMIALS) &&
663         (PrimitiveTrinomialsTable[i][j].degree<d) &&
664         (PrimitiveTrinomialsTable[i][j].degree>0);
665         j++);
666    if ((j==MAX_TRINOMIALS) ||
667        (PrimitiveTrinomialsTable[i][j].degree>d) ||
668        (PrimitiveTrinomialsTable[i][j].degree==0))
669    {
670      for (i=0;
671           (i<POLYNOMIALS_TABLE_SIZE) &&
672           ((PrimitivePolynomialsTable[i].prime<p) ||
673            ((PrimitivePolynomialsTable[i].prime==p) &&
674             (PrimitivePolynomialsTable[i].degree<d)));
675           i++);
676      if ((i==POLYNOMIALS_TABLE_SIZE) ||
677          (PrimitivePolynomialsTable[i].prime>p) ||
678          (PrimitivePolynomialsTable[i].degree>d))
679      {
680        fprintf(stderr,"Error: Primitive polynomial for p=%"PRIuFAST32
681                " and d=%"PRIuFAST8" not found.\n",p,d);
682        exit(EXIT_FAILURE);
683      }
684      for (j=0; (j<MAX_TERMS &&
685                 PrimitivePolynomialsTable[i].terms[j].coefficient>0); j++)
686        if (PrimitivePolynomialsTable[i].terms[j].coefficient>0)
687          vec[PrimitivePolynomialsTable[i].terms[j].degree] =
688            p - PrimitivePolynomialsTable[i].terms[j].coefficient;
689        vec[0] = p - PrimitivePolynomialsTable[i].constantTerm;
690      return;
691    }
692    vec[PrimitiveTrinomialsTable[i][j].aTerm.degree] =
693      p - PrimitiveTrinomialsTable[i][j].aTerm.coefficient;
694    vec[0] = p - PrimitiveTrinomialsTable[i][j].constantTerm;
695  }
696
697
698  void CopyMatrix(matrix dest, matrix src)
699  {
700    CopyData(dest->d, src->d);
701  }
702
703
704  void AddMatrices(matrix dest, matrix src1, matrix src2)
705  {
706    uint_fast16_t pos;
707
```

```
708       for (pos=0; pos < dest->d->length; pos++)
709         dest->d->e[pos] = src1->d->e[pos] + src2->d->e[pos];
710     }
711
712
713     void MultMatrices(matrix dest, matrix src1, matrix src2)
714     {
715       uint_fast8_t r, c, i;
716       uint_fast16_t d_pos, s1_pos, s2_pos;
717       uint64_t v;
718
719       d_pos = 0;
720       s1_pos = 0;
721       for(r=0; r<dest->rows; r++)
722       {
723         for(c=0; c<dest->cols; c++)
724         {
725           s2_pos = c;
726           v = 0;
727           for(i=0; i < (src1->cols); i++)
728           {
729             v += src1->d->e[s1_pos+i] * src2->d->e[s2_pos];
730             s2_pos += src2->cols;
731           }
732           dest->d->e[d_pos++] = v;
733         }
734         s1_pos += src1->cols;
735       }
736     }
737
738
739     void ModMatrix(matrix mat)
740     {
741       uint_fast16_t pos;
742
743       for (pos=0; pos < mat->d->length; pos++)
744        mat->d->e[pos] %= PRIME;
745     }
746
747
748     data CreateData(uint_fast16_t l)
749     {
750       data dat;
751
752       dat = (data)malloc(sizeof(struct_data));
753       dat->length = l;
754       dat->size = l*sizeof(uint64_t);
755       dat->e = (elements)malloc(dat->size);
756       return dat;
757     }
758
759
760     data LinkData(uint_fast16_t l, elements e)
761     {
762       data dat;
763
764       dat = (data)malloc(sizeof(struct_data));
765       dat->length = l;
766       dat->size = l*sizeof(uint64_t);
767       dat->e = e;
768       return dat;
769     }
770
771
772     void CopyData(data dest, data src)
773     {
774       memcpy(dest->e, src->e, dest->size);
775     }
776
777
778     void DeleteData(data dat)
779     {
780       free(dat->e);
781       free(dat);
782     }
```

Universitat d'Alacant
Universidad de Alicante

B.3 Generador óptimo

Las instrucciones para el correcto uso y paso de parámetros, que se muestran a continuación, se obtienen al invocar el ejecutable sin parámetros:

```
Usage: M# l i [file [-q | -t]]

        l:      Sequence bits length (0 = endless)
                It can be followed by B, KB, MB, GB, TB or PB for
                bytes, kilobytes, megabytes, gigabytes or petabytes
        i:      Random key identification (call number)
        file:   Tracker output file name (optional)
        -q:     Quiet tracker (Only arguments and end-begin time)
        -t:     Quiet tracker and no output sequence (Only to test time)
```

M# se refiere a cualquiera de los M7 a M15 y M19, ya que el algoritmo es parametrizable y se puede obtener el ejecutable correspondiente a cualquier valor $d = 7, 8, \ldots, 14, 15, 19$ cambiando en *motor.c* (línea 22) el *#define M10* establecido por defecto en ese fichero a cualquiera de los otros M#.

El código mostrado a continuación puede servir de consulta rápida y aclaración sobre algún punto concreto de la implementación. Para su compilación o reutilización, está disponible el mismo código en el siguiente enlace:

http://www.dccia.ua.es/~fmartine/tesis/M.zip

El enlace incluye un proyecto para el entorno Dev-C++ que facilita la compilación conjunta del módulo motor del generador con el programa.

Universitat d'Alacant
Universidad de Alicante

M.c:

```c
/*
  M#: Pseudorandom number generator based upon 3x3 block upper
    triangular matrix with 32x32 s-boxes filtration and p=65537
    optimized for x^d+(k_5)x+k_4 primitive polynomials
    (Matrices TSB de 3x3 bloques optimizado).

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/


/* headers */

#include <stdlib.h>
#include <stdint.h>
#include <inttypes.h>
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>

#include "motor.h"


/* arguments */
typedef struct
{
  uint64_t l;
  uint_fast8_t i;
  char * fileName;
  uint_fast8_t quiet;
  uint_fast8_t test;
} arguments;

arguments * GetArguments(int argc, char * argv[]);
void DropArguments(arguments * args);
void Usage(char * command);
void PrintArguments(FILE * file, arguments * args);


/* others prototypes */

FILE * OutputFile();
FILE * OpenTrackRegister(arguments * args);
void CloseTrackRegister(FILE * file);
void TimeStamp(FILE * file, char * message);

void GenerateBits(FILE * f, motor m, uint64_t bits);


/* code */

int main(int argc, char * argv[])
{
  arguments * args;
  FILE * file;
  key k;
  LARGE_INTEGER li;
  uint64_t r, f;
  motor m;


  args = GetArguments(argc, argv);
  if (args->fileName != NULL)
    file = OpenTrackRegister(args);
  else
    file = NULL;

  k = CreateKey(args->i);
  if (file)
  {
    if (!args->quiet)
      PrintKey(file, k);
    QueryPerformanceCounter(&li);
    r = (uint64_t)li.QuadPart;
```

```
77    }
78
79    m = CreateMotor(k);
80    GenerateBits((args->test)?NULL:OutputFile(), m, args->l);
81
82    DeleteMotor(m);
83    DeleteKey(k);
84
85    if (file)
86    {
87      if (!args->quiet)
88        PrintMotor(file, m);
89      QueryPerformanceCounter(&li);
90      r = (uint64_t)li.QuadPart - r;
91      QueryPerformanceFrequency(&li);
92      f = li.QuadPart;
93      fprintf(file,"End-Begin Time:\n\tCycles:    %15"PRIu64"\n", r);
94      fprintf(file,"\tFrequency: %15"PRIu64"\n\tSeconds:  %16.8lf\n",
95              f, (double)r/f);
96      CloseTrackRegister(file);
97    }
98    DropArguments(args);
99
100   return EXIT_SUCCESS;
101 }
102
103
104 void GenerateBits(FILE * f, motor m, uint64_t bits)
105 {
106   uint64_t n;
107   uint8_t * bytes;
108   uint_fast16_t size;
109
110   size = SizeOfExtraction(m);
111   bytes = (uint8_t *)malloc(size);
112   n = (bits) ? (bits>>3) : -1;
113   while (n)
114   {
115     GetExtraction(m,bytes);
116     if (n<size) size = n;
117     if (f != NULL)
118       if (!fwrite(bytes,size,1,f))
119       {
120         fprintf(stderr,"Error writing the output file.\n");
121         exit(EXIT_FAILURE);
122       }
123     n-=size;
124   }
125   free(bytes);
126 }
127
128
129 FILE * OutputFile()
130 {
131   if(-1 == _setmode(_fileno(stdout),_O_BINARY))
132     fprintf(stderr,"Cannot set stdout binary mode.\n");
133   return stdout;
134 }
135
136
137 FILE * OpenTrackRegister(arguments * args)
138 {
139   FILE * file;
140
141   file = fopen(args->fileName, "w");
142   if (!file)
143   {
144     fprintf(stderr, "Error opening the tracker file.\n");
145     exit(EXIT_FAILURE);
146   }
147   PrintArguments(file, args);
148   TimeStamp(file, "Begin");
149   return file;
150 }
151
152
153 void CloseTrackRegister(FILE * file)
154 {
155   TimeStamp(file, "End");
```

Universitat d'Alacant
Universidad de Alicante

```
156   fclose(file);
157 }
158
159 void TimeStamp(FILE * file, char * message)
160 {
161   char time_buff[9], date_buff[9];
162
163   if (_strtime_s(time_buff,9) || _strdate_s(date_buff,9))
164   {
165     fprintf(stderr, "Time/Date system error.\n");
166     exit(EXIT_FAILURE);
167   }
168   else
169     fprintf(file, "\n%s: %s %s\n\n", message, date_buff, time_buff);
170 }
171
172
173 arguments * GetArguments(int argc, char * argv[])
174 {
175   arguments * args;
176   char * end;
177   uint8_t i;
178   const uint64_t B = 8;
179   const uint64_t K = 1024 * B;
180   const uint64_t M = 1024 * K;
181   const uint64_t G = 1024 * M;
182   const uint64_t T = 1024 * G;
183   const uint64_t P = 1024 * T;
184
185   args = (arguments *)malloc(sizeof(arguments));
186   if ((argc < 3) || (argc > 5)) Usage(argv[0]);
187
188   i=0;
189   args->l = strtoumax(argv[++i],&end,10);
190   if (!strcmp("B",end)) args->l*=B; else
191     if (!strcmp("KB",end)) args->l*=K; else
192       if (!strcmp("MB",end)) args->l*=M; else
193         if (!strcmp("GB",end)) args->l*=G; else
194           if (!strcmp("TB",end)) args->l*=T; else
195             if (!strcmp("PB",end)) args->l*=P; else
196               if (strlen(end) > 0) Usage(argv[0]);
197   args->i = strtoul(argv[++i],&end,10);
198   if (strlen(end) > 0) Usage(argv[0]);
199
200   if (argc > 3)
201   {
202     args->fileName = argv[++i];
203     if (argc == 5)
204     {
205       i++;
206       if (strcmp("-q",argv[i]) && strcmp("-t",argv[i])) Usage(argv[0]);
207       args->quiet = 1;
208       args->test = !strcmp("-t",argv[i]);
209     }
210     else
211     {
212       args->quiet = 0;
213       args->test = 0;
214     }
215   }
216   else
217   {
218     args->fileName = NULL;
219     args->quiet = 0;
220     args->test = 0;
221   }
222
223   return args;
224 }
225
226
227 void DropArguments(arguments * args)
228 {
229   free(args);
230 }
231
232
233 void Usage(char * command)
234 {
```

Universitat d'Alacant
Universidad de Alicante

```
235   char drive[_MAX_DRIVE];
236   char dir[_MAX_DIR];
237   char fname[_MAX_FNAME];
238   char ext[_MAX_EXT];
239   errno_t err;
240
241   err = _splitpath_s(command,drive,_MAX_DRIVE,dir,_MAX_DIR,
242                      fname,_MAX_FNAME,ext,_MAX_EXT);
243   if (err != 0)
244   {
245     fprintf(stderr,"Error splitting the path. Error code %d.\n", err);
246     exit(EXIT_FAILURE);
247   }
248   fprintf(stderr,"\nUsage: %s l i [file [-q | -t]]\n\n",fname);
249   fprintf(stderr,"\tl:\tSequence bits length (0 = endless)\n");
250   fprintf(stderr,"\t  \tIt can be followed by B, KB, MB, GB, TB or PB for\n");
251   fprintf(stderr,"\t  \tbytes, kilobytes, megabytes, gigabytes or petabytes\n");
252   fprintf(stderr,"\ti:\tRandom key identification (call number)\n");
253   fprintf(stderr,"\tfile:\tTracker output file name (optional)\n");
254   fprintf(stderr,"\t-q:\tQuiet tracker (Only arguments and end-begin time)\n");
255   fprintf(stderr,
256           "\t-t:\tQuiet tracker and no output sequence (Only to test time)\n");
257   fprintf(stderr,"\n");
258   exit(EXIT_FAILURE);
259 }
260
261
262 void PrintArguments(FILE * file, arguments * args)
263 {
264   fprintf(file,"Arguments:\n");
265   if (args->l)
266     fprintf(file,"\tSequence of %"PRIu64" bits\n", args->l);
267   else
268     fprintf(file,"\tEndless sequence of bits\n");
269   fprintf(file,"\tCall identification %"PRIuFAST8". ", args->i);
270   fprintf(file, "Tracker output file: %s%s\n\n",
271           args->fileName, (args->quiet)?" (quiet)":"");
272 }
```

Universitat d'Alacant
Universidad de Alicante

motor.h:

```
/*
  MOTOR: Header of types and functions for pseudorandom
    number generator based upon 3x3 block upper triangular
    matrix with 32x32 s-boxes filtration and p=65537.
  Version: Optimized for x^d+(k_5)x+k_4 primitive polynomials.

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/

#ifndef _MOTOR_H
#define _MOTOR_H

/* headers */

#include <stdint.h>
#include <stdio.h>


/* key */
typedef uint8_t * key;
#define KEYBYTESLENGTH 256

key CreateKey(uint_fast8_t id);
void DeleteKey(key k);
void PrintKey(FILE * file, key k);

/* motor */

#ifndef _STRUCT_MOTOR
#define _STRUCT_MOTOR
#endif

typedef _STRUCT_MOTOR * motor; /* Pointer a private structure */

motor CreateMotor(key k);
void DeleteMotor(motor m);
void PrintMotor(FILE * file, motor m);
uint_fast16_t SizeOfExtraction(motor m);
void GetExtraction(motor m, uint8_t * bytes);

#endif  /* _MOTOR_H */
```

Universitat d'Alacant
Universidad de Alicante

motor.c:

```
/*
  MOTOR: Implementation module of functions for pseudorandom
    number generator based upon 3x3 block upper triangular
    matrix with 32x32 s-boxes filtration and p=65537.
  Version: Optimized for x^d+(k_5)x+k_4 primitive polynomials.

  Author: Francisco Miguel Martinez Perez.
  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License.
  (CC) BY-NC-SA 2015
*/

#define _MOTOR_H

/* headers */

#include <stdint.h>
#include <inttypes.h>
#include <stdlib.h>
#include <string.h>

/* choice a size (M7, ..., M15, M19)*/
#define M10

/* define section */

#define K0 65537    /* K0=65537    is p for Zp                  (r=1 s=2 t=d) */
#define K1 3        /* K1=3        is primitive root Zp p=65537          (r=1) */
#define K2 65534    /* For K2 & K3 is used primitive polinomial x^2+x+3 (s=2) */
#define K3 65536    /* K2=K0-3=65534           K3=K0-1=65536                  */

#ifdef M7
#define D 7         /* Size of block matrix C: d=t=7                    (d=7) */
#define K4 65534    /* For K4 & K5 is used primitive polinomial x^7+x+3       */
#define K5 65536    /* K4=K0-3=65534           K5=K0-1=65536                  */
#define LENGTH 23   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=23                     */
#define ODD
#endif
#ifdef M8
#define D 8         /* Size of block matrix C: d=t=8                    (d=8) */
#define K4 65517    /* For K4 & K5 is used primitive polinomial x^8+x+20      */
#define K5 65536    /* K4=K0-20=65517          K5=K0-1=65536                  */
#define LENGTH 26   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=26                     */
#endif
#ifdef M9
#define D 9         /* Size of block matrix C: d=t=9                    (d=9) */
#define K4 65527    /* For K4 & K5 is used primitive polinomial x^9+x+10      */
#define K5 65536    /* K4=K0-10=65527          K5=K0-1=65536                  */
#define LENGTH 29   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=29                     */
#define ODD
#endif
#ifdef M10
#define D 10        /* Size of block matrix C: d=t=10                  (d=10) */
#define K4 65531    /* For K4 & K5 is used primitive polinomial x^10+x+6      */
#define K5 65536    /* K4=K0-6=65531           K5=K0-1=65536                  */
#define LENGTH 32   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=32                     */
#endif
#ifdef M11
#define D 11        /* Size of block matrix C: d=t=11                  (d=11) */
#define K4 65510    /* For K4 & K5 is used primitive polinomial x^11+x+27     */
#define K5 65536    /* K4=K0-27=65510          K5=K0-1=65536                  */
#define LENGTH 35   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=35                     */
#define ODD
#endif
#ifdef M12
#define D 12        /* Size of block matrix C: d=t=12                  (d=12) */
#define K4 65526    /* For K4 & K5 is used primitive polinomial x^12+x+11     */
#define K5 65536    /* K4=K0-11=65526          K5=K0-1=65536                  */
#define LENGTH 38   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=38                     */
#endif
#ifdef M13
#define D 13        /* Size of block matrix C: d=t=13                  (d=13) */
#define K4 65481    /* For K4 & K5 is used primitive polinomial x^13+x+56     */
#define K5 65536    /* K4=K0-56=65481          K5=K0-1=65536                  */
#define LENGTH 41   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=41                     */
#define ODD
```

```
76  #endif
77  #ifdef M14
78  #define D 14        /* Size of block matrix C: d=t=14                  (d=14) */
79  #define K4 65429    /* For K4 & K5 is used primitive polinomial x^14+x+108    */
80  #define K5 65536    /* K4=KO-108=65429          K5=KO-1=65536                 */
81  #define LENGTH 44   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=44                     */
82  #endif
83  #ifdef M15
84  #define D 15        /* Size of block matrix C: d=t=15                  (d=15) */
85  #define K4 65517    /* For K4 & K5 is used primitive polinomial x^15+x+20     */
86  #define K5 65536    /* K4=KO-20=65517           K5=KO-1=65536                 */
87  #define LENGTH 47   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=47                     */
88  #define ODD
89  #endif
90  #ifdef M19
91  #define D 19        /* Size of block matrix C: d=t=19                  (d=19) */
92  #define K4 65490    /* For K4 & K5 is used primitive polinomial x^19+x+47     */
93  #define K5 65536    /* K4=KO-47=65490           K5=KO-1=65536                 */
94  #define LENGTH 59   /* LENGTH=r*s+r*t+s*t=2+t+2*t=2+3d=44                     */
95  #define ODD
96  #endif
97
98
99
100 /* types */
101
102 /* data */
103 typedef uint64_t data[LENGTH];
104
105 void CopyData(data dest, data src);
106
107 /* s-boxes */
108 #define SB_ELEMENTS 256
109 #define SB_SBOXES 4
110 typedef uint8_t sbox8x8[SB_ELEMENTS];
111 typedef uint32_t sbox8x32[SB_ELEMENTS];
112 typedef sbox8x32 sboxes[SB_SBOXES];
113
114 /* motor */
115 #define N_VAR 5     /* r^2+s^2=1+4=5 */
116 typedef uint64_t state[N_VAR];
117 typedef struct
118 {
119   sboxes sbox;
120   uint32_t filter;
121   data S;  /* Seed       */
122   data P;  /* Production */
123   state V; /* Variables  */
124 } struct_motor;
125
126 #define _STRUCT_MOTOR struct_motor
127 #undef _MOTOR_H
128 #include "motor.h"
129
130 void Iterate(data S, data P, state V);
131
132 void CreateSboxes(sboxes sbox, key k);
133
134 void FillSeed(data S, sboxes sbox, key k);
135
136
137
138
139 /* Code */
140
141
142 key CreateKey(uint_fast8_t id)
143 {
144   key k;
145   uint_fast16_t i;
146
147   k = (key)malloc(sizeof(uint8_t)*KEYBYTESLENGTH);
148
149   srand(101*id);
150   for (i=0; i < KEYBYTESLENGTH; i++) k[i] = rand() % 256;
151
152   return k;
153 }
154
```

Universitat d'Alacant
Universidad de Alicante

```
155
156   void DeleteKey(key k)
157   {
158     free(k);
159   }
160
161
162   void PrintKey(FILE * file, key k)
163   {
164     uint_fast16_t i;
165
166     fprintf(file, "Key:");
167     for (i=0; i < KEYBYTESLENGTH; i++)
168         fprintf(file, "%s %03"PRIx8"", (i%20)?"":"\n", k[i]);
169     fprintf(file, "\n");
170   }
171
172
173   motor CreateMotor(key k)
174   {
175     motor m;
176     uint8_t i;
177
178     m = (motor)malloc(sizeof(struct_motor));
179
180     CreateSboxes(m->sbox, k);
181     m->filter = 0;
182
183     FillSeed(m->S, m->sbox, k);
184
185     CopyData(m->P, m->S);
186     m->V[0] = K1;
187     m->V[1] = 0;
188     m->V[2] = 1;
189     m->V[3] = K2;
190     m->V[4] = K3;
191
192     for (i=0; i < 64; i++)
193       Iterate(m->S, m->P, m->V);
194
195     return m;
196   }
197
198
199   void DeleteMotor(motor m)
200   {
201     free(m);
202   }
203
204
205   void PrintMotor(FILE * file, motor m)
206   {
207     uint_fast8_t i, j;
208
209     i = 0;
210     fprintf(file, "M:\n[");
211     fprintf(file,"%6"PRIu64"] [",K1);
212     fprintf(file,"%6"PRIu64"%6"PRIu64"] [",m->S[i++],m->S[i++]);
213     for(j=0; j<((LENGTH-2)/3); j++)
214       fprintf(file,"%6"PRIu64"",m->S[i++]);
215     fprintf(file,"]\n          [     0     1] [");
216     for(j=0; j<((LENGTH-2)/3); j++)
217       fprintf(file,"%6"PRIu64"",m->S[i++]);
218     fprintf(file,"]\n          [");
219     fprintf(file,"%6"PRIu64"%6"PRIu64"] [",K2,K3);
220     for(j=0; j<((LENGTH-2)/3); j++)
221       fprintf(file,"%6"PRIu64"",m->S[i++]);
222     i = 0;
223     fprintf(file, "]\nM^h:\n[");
224     fprintf(file,"%6"PRIu64"] [",m->V[0]);
225     fprintf(file,"%6"PRIu64"%6"PRIu64"] [",m->P[i++],m->P[i++]);
226     for(j=0; j<((LENGTH-2)/3); j++)
227       fprintf(file,"%6"PRIu64"",m->P[i++]);
228     fprintf(file,"]\n          [");
229     fprintf(file,"%6"PRIu64"%6"PRIu64"] [",m->V[1],m->V[2]);
230     for(j=0; j<((LENGTH-2)/3); j++)
231       fprintf(file,"%6"PRIu64"",m->P[i++]);
232     fprintf(file,"]\n          [");
233     fprintf(file,"%6"PRIu64"%6"PRIu64"] [",m->V[3],m->V[4]);
```

Universitat d'Alacant
Universidad de Alicante

```
234       for(j=0; j<((LENGTH-2)/3); j++)
235         fprintf(file,"%6"PRIu64"",m->P[i++]);
236       fprintf(file, "]\n");
237     }
238
239
240     uint_fast16_t SizeOfExtraction(motor m)
241     {
242     #ifndef ODD
243       return LENGTH * 2;
244     #else
245       return LENGTH * 4;
246     #endif
247     }
248
249
250     void GetExtraction(motor m, uint8_t * bytes)
251     {
252       uint_fast16_t i, j, k;
253
254       j=0;
255       Iterate(m->S, m->P, m->V); /* Iteration. */
256       i=0;
257       for (k=0; k < LENGTH/2; k++)
258       {              /* Filtration with s-box. */
259         m->filter ^= m->sbox[0][(uint8_t)((m->P[i])>>8)];
260         m->filter ^= m->sbox[1][(uint8_t)(m->P[i++])];
261         m->filter ^= m->sbox[2][(uint8_t)((m->P[i])>>8)];
262         m->filter ^= m->sbox[3][(uint8_t)(m->P[i++])];
263         bytes[j++] = m->filter;
264         bytes[j++] = (m->filter)>>8;
265         bytes[j++] = (m->filter)>>16;
266         bytes[j++] = (m->filter)>>24;
267       }
268     #ifdef ODD
269       m->filter ^= m->sbox[0][(uint8_t)((m->P[i])>>8)]; /* Last element of... */
270       m->filter ^= m->sbox[1][(uint8_t)(m->P[i++])];    /* first iterarion.   */
271       Iterate(m->S, m->P, m->V);     /* Second iteration... */
272       i=0;                           /* for odd lengths     */
273       m->filter ^= m->sbox[2][(uint8_t)((m->P[i])>>8)]; /* First element of... */
274       m->filter ^= m->sbox[3][(uint8_t)(m->P[i++])];    /* second iterarion.   */
275       bytes[j++] = m->filter;
276       bytes[j++] = (m->filter)>>8;  /* Filtration with s-box of last...   */
277       bytes[j++] = (m->filter)>>16; /* element of first iteration and...  */
278       bytes[j++] = (m->filter)>>24; /* first element of second iterarion. */
279       for (k=0; k < LENGTH/2; k++)
280       {                  /* Filtration with s-box of remainder elements. */
281         m->filter ^= m->sbox[0][(uint8_t)((m->P[i])>>8)];
282         m->filter ^= m->sbox[1][(uint8_t)(m->P[i++])];
283         m->filter ^= m->sbox[2][(uint8_t)((m->P[i])>>8)];
284         m->filter ^= m->sbox[3][(uint8_t)(m->P[i++])];
285         bytes[j++] = m->filter;
286         bytes[j++] = (m->filter)>>8;
287         bytes[j++] = (m->filter)>>16;
288         bytes[j++] = (m->filter)>>24;
289       }
290     #endif
291
292     }
293
294
295     void Iterate(data S, data P, state V)
296     {
297
298       data A; /* Auxiliar */
299       uint_fast8_t i, j;
300
301       /* Almacenamiento auxiliar de P en A */
302       CopyData(A, P);
303
304       /* Nuevo P calculado de Ks, S, V y A */
305       P[0] = ((V[0]*S[0])+(A[1]*K2))%K0;
306       P[1] = ((V[0]*S[1])+(A[0]+(A[1]*K3)))%K0;
307
308       j = 1 + D;
309       P[2] = (((V[0]*S[2])+((A[0]*S[j+1])+(A[1]*S[2*j])))+(A[j]*K4))%K0;
310       P[3] = (((V[0]*S[3])+((A[0]*S[j+2])+(A[1]*S[2*j+1])))+(A[2]+(A[j]*K5)))%K0;
311       for (i=4; i <= j; i++)
312         P[i] = (((V[0]*S[i])+((A[0]*S[i+D])+(A[1]*S[i+2*D])))+A[i-1])%K0;
```

```
313
314       j += D;
315       P[i] = (((V[1]*S[i])+(V[2]*S[j+1]))+(A[j]*K4))%K0;
316       i++;
317       P[i] = (((V[1]*S[i])+(V[2]*S[j+2]))+(A[i-1]+(A[j]*K5)))%K0;
318       for (i++; i <= j; i++)
319         P[i] = (((V[1]*S[i])+(V[2]*S[i+D]))+A[i-1])%K0;
320
321       j += D;
322       P[i] = (((V[3]*S[i-D])+(V[4]*S[i]))+(A[j]*K4))%K0;
323       i++;
324       P[i] = (((V[3]*S[i-D])+(V[4]*S[i]))+(A[i-1]+(A[j]*K5)))%K0;
325       for (i++; i <= j; i++)
326         P[i] = (((V[3]*S[i-D])+(V[4]*S[i]))+A[i-1])%K0;
327
328       /* Almacenamiento auxiliar de V en A */
329       A[0] = V[1];
330       A[1] = V[3];
331
332       /* Nuevo V calculado de Ks, V y A */
333       V[0] = (V[0]*K1)%K0;
334       V[1] = (V[2]*K2)%K0;
335       V[2] = (A[0]+(V[2]*K3))%K0;
336       V[3] = (V[4]*K2)%K0;
337       V[4] = (A[1]+(V[4]*K3))%K0;
338     }
339
340
341     void CreateSboxes(sboxes sbox, key k)
342     {
343       sbox8x8 s;
344       uint_fast16_t i, h;
345       uint_fast8_t j, c;
346       uint32_t t;
347
348       for (i=0; i < SB_ELEMENTS; i++)
349       {
350         s[i] = i;
351         for (j=0; j < SB_SBOXES; j++)
352           sbox[j][i] = 0;
353       }
354
355       for (j=0; j < SB_SBOXES; j++)
356         for (c=0; c < 4; c++)
357         {
358           h=0;
359           for (i=0; i < SB_ELEMENTS; i++)
360           {
361             h = (h + s[i] + k[i%KEYBYTESLENGTH]) % SB_ELEMENTS;
362             t = s[i];
363             s[i] = s[h];
364             s[h] = t;
365           }
366           for (i=0; i < SB_ELEMENTS; i++)
367             sbox[j][i] = (sbox[j][i]<<8) ^ s[i];
368         }
369     }
370
371
372     void FillSeed(data S, sboxes sbox, key k)
373     {
374       uint_fast16_t i, j;
375       uint64_t v;
376
377       j = 0;
378       S[0] = 0x5A5A;
379       for (i=1; i < LENGTH-1; i++)
380       {
381         v = sbox[0][(i+k[j%16])%256];
382         j++;
383         v ^= sbox[1][(j+k[j%16])%256];
384         j++;
385         v<<=32;
386         v ^= sbox[2][(i+k[j%16])%256];
387         j++;
388         v ^= sbox[3][(j+k[j%16])%256];
389         j++;
390         S[i] = v % K0;
391       }
```

Universitat d'Alacant
Universidad de Alicante

```
392       S[i] = 0xA5A5;
393     }
394
395     void CopyData(data dest, data src)
396     {
397       memcpy(dest, src, LENGTH*sizeof(uint64_t));
398     }
```

Universitat d'Alacant
Universidad de Alicante

Apéndice C
Polinomios primitivos

A continuación se muestran las tablas de polinomios primitivos y generadores (raíces primitivas) usadas para las baterías de pruebas.

En las tablas se dispone de muchos polinomios primitivos que, si bien no se han utilizado todos, han ayudado a diseñar estas baterías en función de su disponibilidad.

Para el grado 7 en los primos 173, 271 y 293 y el grado 6 en el primo 257, la tabla no dispone de polinomio primitivo, así como para el primo 17 y grado 59. Estas carencias han supuesto, como se indica en los correspondientes apartados del capítulo 3, que en algunos casos se haya tenido que renunciar a algunas pruebas y en otros se hayan tenido que sustituir por otras similares.

Universitat d'Alacant
Universidad de Alicante

Los polinomios de las tablas se han recopilado gracias al uso del servidor web PIPS (*Primitive and Irreducible Polynomial Server*) [35] y las librerías Primpoly Version 12.0 ("A Program for Computing Primitive Polynomials") [75].

Para las primeras tandas de experimentos, casi la totalidad de los polinomios usados han sido obtenidos del servidor web implementado como ejemplo de uso de la librería ZEN [36] (una herramienta para el cálculo rápido en extensiones y anillos finitos). Este servidor admite búsquedas de polinomios primitivos con la posibilidad de solicitar, en concreto, un trinomio.

Las tablas pueden servir para una consulta rápida o estimación visual de los polinomios disponibles. Para comodidad en su tratamiento se dispone de una hoja de cálculo con las mismas tablas en el siguiente enlace:

http://www.dccia.ua.es/~fmartine/tesis/TablasPrimitivos.zip

Los trinomios se han remarcado con (*) y los que, además, tienen el coeficiente de su término en x no nulo, con un (**).

Polinomios primitivos para el primo **2**

(**) x^2+x+1
(**) x^3+x+1
(**) x^4+x+1
(*) x^5+x^2+1
(*) x^5+x^3+1
(**) x^6+x+1
(*) x^7+x^4+1
(**) x^7+x+1
$x^8+x^4+x^3+x^2+1$
$x^8+x^6+x^5+x^3+1$
(*) x^9+x^4+1
(*) $x^{10}+x^3+1$
(*) $x^{11}+x^2+1$
$x^{12}+x^6+x^4+x+1$
$x^{13}+x^4+x^3+x+1$
$x^{14}+x^5+x^3+x+1$
(**) $x^{15}+x+1$
$x^{16}+x^5+x^3+x^2+1$
(*) $x^{17}+x^3+1$
$x^{18}+x^5+x^2+x+1$
$x^{19}+x^5+x^2+x+1$
(*) $x^{20}+x^3+1$
(*) $x^{21}+x^2+1$
(**) $x^{22}+x+1$
(*) $x^{23}+x^5+1$
$x^{24}+x^4+x^3+x+1$
(*) $x^{25}+x^3+1$
$x^{26}+x^6+x^2+x+1$
$x^{27}+x^5+x^2+x+1$
(*) $x^{28}+x^3+1$
(*) $x^{29}+x^2+1$
$x^{30}+x^6+x^4+x+1$
(*) $x^{31}+x^3+1$
$x^{32}+x^7+x^5+x^3+x^2+x+1$
$x^{33}+x^6+x^4+x+1$
$x^{34}+x^7+x^6+x^5+x^2+x+1$
(*) $x^{35}+x^2+1$
$x^{36}+x^6+x^5+x^4+x^2+x+1$
$x^{37}+x^5+x^4+x^3+x^2+x+1$
$x^{38}+x^6+x^5+x+1$
(*) $x^{39}+x^4+1$
$x^{40}+x^5+x^4+x^3+1$
(*) $x^{41}+x^3+1$
$x^{42}+x^5+x^4+x^3+x^2+x+1$
$x^{43}+x^6+x^4+x^3+1$
$x^{44}+x^6+x^5+x^2+1$
$x^{45}+x^4+x^3+x+1$
$x^{46}+x^8+x^5+x^3+x^2+x+1$
(*) $x^{47}+x^5+1$
$x^{48}+x^7+x^5+x^4+x^2+x+1$
$x^{49}+x^6+x^5+x^4+1$
$x^{50}+x^4+x^3+x^2+1$
$x^{51}+x^6+x^3+x+1$
(*) $x^{52}+x^3+1$
$x^{53}+x^6+x^2+x+1$
$x^{54}+x^6+x^5+x^4+x^3+x^2+1$

Polinomios primitivos para el primo **2**

$x^{55}+x^6+x^2+x+1$
$x^{56}+x^7+x^4+x^2+1$
$x^{57}+x^5+x^3+x^2+1$
$x^{58}+x^6+x^5+x+1$
$x^{59}+x^6+x^5+x^4+x^3+x+1$
(**) $x^{60}+x+1$
$x^{61}+x^5+x^2+x+1$
$x^{62}+x^6+x^5+x^3+1$
(**) $x^{63}+x+1$
$x^{64}+x^4+x^3+x+1$
$x^{65}+x^4+x^3+x+1$
$x^{66}+x^8+x^6+x^5+x^3+x^2+1$
$x^{67}+x^5+x^2+x+1$
$x^{68}+x^7+x^5+x+1$
$x^{69}+x^6+x^5+x^2+1$

Polinomios primitivos para el primo **3**

(**) x^2+x+2
(**) x^3+2x+1
(*) x^4+x^3+2
(**) x^5+2x+1
(**) x^6+2x+2
(*) x^7+2x^2+1
(*) x^8+x^5+2
(*) x^9+2x^4+1

Polinomios primitivos para el primo **5**

(**) x^2+4x+2
(*) x^2+x^2+2
(**) x^2+x+2
(*) x^3+3x^2+2
(**) x^3+3x+2
x^4+4x^2+4x+3
x^4+x^2+2x+2
(*) x^5+3x^3+2
(**) x^5+4x+2
(**) x^6+2x+3
(**) x^6+x+2
(**) x^7+3x+2
(*) x^7+4x^3+3
$x^8+3x^7+4x^5+x^4+2x^3+4x^2+x+3$
x^8+x^2+2x+3
(*) x^9+3x^4+2
x^9+x^2+2x+3
$x^{10}+x^2+x+3$
(**) $x^{11}+3x+2$
$x^{12}+x^3+2x+3$
$x^{13}+x^2+3x+2$
$x^{14}+x^3+x^2+2x+2$
(*) $x^{15}+x^2+2$
$x^{16}+x^3+3x+2$
$x^{17}+2x^2+2x+2$
$x^{18}+x^4+2x+2$
$x^{19}+x^3+2x+2$
$x^{20}+x^2+2x+3$
(**) $x^{21}+4x+2$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **5**

$x^{22}+x^2+x+3$
$x^{23}+x^2+4x+3$
$x^{24}+x^3+x^2+x+2$
$x^{25}+2x^3+3x+2$
$x^{26}+x^3+x^2+2x+2$
(**) $x^{27}+4x+2$
$x^{28}+x^3+2x+3$
$x^{29}+x^3+x^2+3$
$x^{30}+x^4+2x^2+x+3$
(**) $x^{31}+3x+2$
$x^{32}+x^3+x^2+4x+2$
$x^{33}+2x^2+2x+2$
$x^{35}+x^3+x^2+3x+2$
$x^{36}+x^3+3x+2$
$x^{37}+x^2+3x+2$
$x^{38}+x^3+3x^2+x+2$
$x^{39}+2x^3+3x+2$
$x^{40}+x^3+x^2+3x+3$
(*) $x^{41}+4x^3+2$
$x^{42}+x^3+3x^2+x+2$
(**) $x^{43}+3x+2$
$x^{45}+2x^2+2x+2$
$x^{46}+x^4+2x^3+x^2+3$
$x^{47}+x^4+x^3+x^2+2$
$x^{48}+x^4+x^3+2x+2$
$x^{49}+x^3+x^2+4x+2$
$x^{50}+x^4+x^3+2x^2+3$
$x^{51}+x^3+x^2+3$
$x^{52}+x^4+x^3+x^2+2x+3$
$x^{53}+2x^3+2x^2+x+3$
$x^{55}+3x^3+x+2$
$x^{56}+x^3+x^2+4x+2$
(**) $x^{57}+4x+2$
$x^{58}+x^4+2x^2+2x+2$
$x^{59}+x^3+2x^2+x+3$
$x^{60}+x^3+x^2+3x+3$
$x^{61}+x^2+2x+3$
$x^{62}+x^4+x^3+x^2+x+2$
(*) $x^{63}+x^2+2$
$x^{65}+3x^3+x^2+x+2$
$x^{66}+2x^4+4x^2+x+3$
$x^{67}+x^4+x^3+x+3$
$x^{68}+x^3+3x^2+3$
$x^{69}+x^3+2x^2+3x+2$

Polinomios primitivos para el primo **7**

(**) x^2+4x+5
(**) x^2+x+3
(**) x^3+3x+2
(*) x^3+6x^2+4
$x^4+3x^3+5x^2+2x+5$
x^4+x^2+3x+5
(*) x^5+5x^3+4
(**) x^5+x+4
x^6+3x^2+x+5
$x^6+x^5+2x^3+3x^2+5x+5$

Polinomios primitivos para el primo **7**

(*) x^7+2x^5+2
(**) x^7+6x+2
(**) x^8+2x+3
(**) x^8+x+3
(*) x^9+3x^7+2
x^9+x^2+x+2
$x^{10}+5x^2+x+5$
(**) $x^{11}+x+4$
$x^{12}+3x^2+2x+3$
$x^{13}+x^2+x+2$
$x^{14}+2x^2+2x+3$
$x^{15}+3x^2+3x+4$
(**) $x^{16}+2x+3$
(**) $x^{17}+x+4$
$x^{18}+3x^2+x+5$
$x^{19}+2x^2+2x+4$
$x^{20}+4x^2+x+5$
$x^{21}+3x^2+3x+4$
(*) $x^{22}+x^3+3$
$x^{23}+4x^2+x+2$
$x^{24}+x^3+3x^2+3x+3$
$x^{25}+x^3+4x+2$
$x^{26}+x^2+x+3$
$x^{27}+x^2+2x+4$
$x^{28}+5x^2+2x+5$
(**) $x^{29}+5x+2$
$x^{30}+x^2+x+5$
$x^{31}+x^2+4x+2$
$x^{32}+x^3+6x+5$
$x^{33}+x^2+5x+4$
(**) $x^{34}+2x+3$
$x^{35}+5x^2+2x+4$
$x^{36}+x^3+x^2+4x+5$
(**) $x^{37}+6x+2$
$x^{38}+x^2+x+3$
$x^{39}+x^2+x+2$
(**) $x^{40}+2x+3$
$x^{41}+x^3+2x^2+5x+2$
$x^{42}+2x^3+x^2+6x+3$
$x^{44}+x^3+6x+3$
$x^{45}+2x^3+x^2+4x+2$
$x^{48}+x^4+x^3+5x^2+5x+5$
$x^{49}+x^3+4x+2$
$x^{51}+x^2+2x+4$
$x^{52}+5x^2+x+5$
(**) $x^{53}+5x+2$
$x^{54}+2x^3+3x^2+x+3$
$x^{55}+3x^2+x+4$
$x^{56}+x^2+x+5$
(**) $x^{57}+3x+2$
$x^{59}+x^3+x^2+2x+4$
$x^{60}+2x^3+x^2+3x+5$
$x^{61}+5x^2+5x+2$
$x^{62}+x^3+5x^2+5x+5$
$x^{63}+x^4+2x^2+2$
(**) $x^{64}+2x+3$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **7**

$x^{66} + 2x^3 + x^2 + 5$
$x^{67} + 3x^2 + 2x + 2$

Polinomios primitivos para el primo **11**

(**) $x^2 + 6x + 2$
(**) $x^3 + 3x + 9$
(*) $x^4 + 4x^3 + 2$
(*) $x^5 + 2x^2 + 9$
$x^6 + 10x + 5x^2 + x^3 + 8x^4 + 9x^5 + 7$
(**) $x^7 + 2x + 5$
$x^8 + 4x^7 + 6x^6 + 2x^5 + x^4 + 7x^3 + x^2 + x + 8$
(*) $x^9 + 2x^8 + 3$

Polinomios primitivos para el primo **13**

(**) $x^2 + 6x + 2$
(*) $x^3 + 3x^2 + 2$
$x^4 + 5x^3 + 7x^2 + 6x + 6$
(*) $x^5 + 8x^3 + 6$
$x^6 + 2x^5 + 9x^3 + 5x^2 + 10x + 6$
(**) $x^7 + 3x + 6$
$x^8 + 3x^7 + 3x^6 + 5x^5 + 10x^4 + 6x^3 + 10x^2 + 2$
(*) $x^9 + 4x^8 + 7$

Polinomios primitivos para el primo **17**

(**) $x^2 + 6x + 6$
(**) $x^2 + x + 3$
(**) $x^3 + 15x + 10$
(**) $x^3 + x + 3$
(**) $x^4 + 15x + 6$
(**) $x^4 + x + 11$
(*) $x^5 + 3x^4 + 14$
(**) $x^5 + x + 3$
(*) $x^6 + 3x^5 + 11$
(**) $x^6 + x + 12$
(**) $x^7 + 13x + 6$
(**) $x^7 + x + 5$
$x^8 + x^2 + 3x + 3$
(**) $x^9 + 7x + 3$
(**) $x^{10} + x + 7$
(**) $x^{11} + 3x + 7$
$x^{12} + 3x^2 + 2x + 5$
(**) $x^{13} + 2x + 6$
$x^{14} + x^2 + 2x + 6$
(**) $x^{15} + 3x + 6$
$x^{16} + x^2 + 4x + 10$
(**) $x^{17} + 16x + 3$
(**) $x^{18} + x + 3$
(**) $x^{19} + 3x + 6$
$x^{20} + x^2 + x + 3$
$x^{21} + x^2 + 12x + 6$
$x^{22} + x^2 + 2x + 5$
$x^{23} + x^2 + 13x + 11$
$x^{24} + x^2 + 2x + 7$
(**) $x^{25} + x + 3$
$x^{26} + x^3 + 13x + 3$
(*) $x^{27} + x^2 + 7$

Polinomios primitivos para el primo **17**

$x^{28} + x^2 + x + 11$
(*) $x^{29} + x^2 + 10$
$x^{30} + x^3 + 10x + 11$
(**) $x^{31} + x + 7$
$x^{32} + x^2 + x + 6$
(*) $x^{33} + x^2 + 6$
$x^{34} + 5x^2 + x + 3$
$x^{35} + x^2 + 3x + 7$
$x^{36} + x^3 + 8x + 6$
$x^{38} + 2x^2 + x + 6$
$x^{39} + x^2 + 5x + 7$
(**) $x^{40} + x + 5$
$x^{42} + x^3 + 10x + 12$
$x^{44} + x^2 + 8x + 12$
$x^{45} + x^2 + x + 7$
$x^{47} + x^2 + 8x + 3$
$x^{48} + x^2 + 2x + 6$
$x^{50} + 2x^2 + x + 14$
$x^{52} + x^2 + 7x + 7$
$x^{56} + x^3 + 2x + 5$
$x^{58} + 3x^2 + 3x + 11$
(**) $x^{61} + 2x + 6$
$x^{64} + 3x^2 + 3x + 3$
$x^{66} + x^3 + 16x + 11$
$x^{67} + x^2 + 11x + 3$

Polinomios primitivos para el primo **19**

(**) $x^2 + 10x + 3$
(*) $x^3 + 5x^2 + 9$
(**) $x^4 + 9x + 10$
(*) $x^5 + 8x^4 + 16$
(**) $x^6 + 10x + 2$
(*) $x^7 + 14x^2 + 16$

Polinomios primitivos para el primo **23**

(**) $x^2 + 3x + 11$
(**) $x^3 + 7x + 4$
(**) $x^4 + 15x + 15$
(*) $x^5 + 6x^4 + 12$
(*) $x^6 + 17x^5 + 15$
(*) $x^7 + 13x^5 + 12$

Polinomios primitivos para el primo **29**

(**) $x^2 + 12x + 21$
(*) $x^3 + 17x^2 + 18$
(*) $x^4 + 6x^3 + 10$
(*) $x^5 + 2x^3 + 27$
(*) $x^6 + 8x^5 + 27$
(*) $x^7 + 13x^5 + 18$

Polinomios primitivos para el primo **31**

(**) $x^2 + 28x + 12$
(**) $x^2 + x + 12$
(**) $x^3 + 13x + 20$
(**) $x^3 + x + 14$
(**) $x^4 + 11x + 21$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **31**

(**) $x^4 + 2x + 17$
(*) $x^5 + 13x^4 + 18$
(**) $x^5 + x + 18$
(*) $x^6 + 15x^5 + 22$
(**) $x^6 + 2x + 3$
(*) $x^7 + 26x^5 + 10$
(**) $x^7 + x + 18$
(**) $x^8 + x + 22$
(**) $x^9 + x + 10$
(**) $x^{10} + x + 11$
(**) $x^{11} + 7x + 9$
$x^{12} + 2x^2 + 3x + 22$
(**) $x^{13} + 5x + 7$
(**) $x^{14} + x + 12$
(*) $x^{15} + x^2 + 9$
$x^{16} + x^2 + 8x + 22$
(**) $x^{17} + 3x + 19$
(**) $x^{18} + x + 3$
(**) $x^{19} + x + 7$
$x^{20} + x^2 + x + 3$
(**) $x^{21} + 4x + 7$
(**) $x^{22} + 5x + 3$
(**) $x^{23} + 6x + 18$
$x^{24} + x^2 + 8x + 3$
(**) $x^{25} + 3x + 10$
(**) $x^{26} + x + 3$
$x^{28} + x^2 + 6x + 11$
(**) $x^{29} + x + 10$
$x^{30} + x^2 + 6x + 22$
(**) $x^{31} + 30x + 7$
(**) $x^{32} + x + 12$
(*) $x^{33} + x^2 + 20$
(**) $x^{35} + 3x + 19$
$x^{36} + x^2 + 9x + 3$
$x^{39} + x^2 + 3x + 19$
$x^{44} + x^2 + 4x + 21$
$x^{45} + x^2 + x + 14$
$x^{49} + x^2 + 4x + 9$
$x^{55} + x^2 + 12x + 18$

Polinomios primitivos para el primo **37**

(**) $x^2 + 31x + 15$
(**) $x^3 + 28x + 32$
(**) $x^4 + 11x + 2$
(*) $x^5 + 24x^3 + 5$
(**) $x^6 + 13x + 22$
(*) $x^7 + 25x^2 + 32$

Polinomios primitivos para el primo **41**

(**) $x^2 + 24x + 15$
(*) $x^3 + 7x^2 + 7$
(**) $x^4 + 18x + 12$
(*) $x^5 + 11x^2 + 17$
(*) $x^6 + 18x^5 + 15$
(*) $x^7 + 9x^4 + 26$

Polinomios primitivos para el primo **43**

(**) $x^2 + 5x + 5$
(*) $x^3 + 33x^2 + 23$
(**) $x^4 + 22x + 29$
(*) $x^5 + 27x^3 + 13$
(**) $x^6 + 32x + 30$
(*) $x^7 + 30x^2 + 38$

Polinomios primitivos para el primo **47**

(**) $x^2 + 31x + 44$
(**) $x^3 + 42x + 32$
(**) $x^4 + 10x + 45$
(*) $x^5 + 5x^4 + 17$
(*) $x^6 + 38x^5 + 41$
(*) $x^7 + 14x^6 + 12$

Polinomios primitivos para el primo **53**

(**) $x^2 + 26x + 18$
(*) $x^3 + 38x^2 + 22$
(**) $x^4 + 15x + 27$
(**) $x^5 + 48x + 21$
(*) $x^6 + 31x^5 + 32$
(*) $x^7 + 18x^2 + 5$

Polinomios primitivos para el primo **59**

(**) $x^2 + 14x + 50$
(**) $x^3 + 27x + 51$
(**) $x^4 + 58x + 14$
(**) $x^5 + 39x + 27$
(**) $x^6 + 37x + 55$
(*) $x^7 + 22x^2 + 26$

Polinomios primitivos para el primo **61**

(**) $x^2 + 44x + 26$
(**) $x^2 + x + 2$
(**) $x^3 + 12x + 6$
(**) $x^3 + x + 17$
(**) $x^4 + 39x + 55$
(**) $x^4 + x + 2$
(*) $x^5 + 19x^4 + 18$
(**) $x^5 + x + 7$
(*) $x^6 + 23x^5 + 7$
(**) $x^6 + 4x + 10$
(**) $x^7 + 2x + 2$
(*) $x^7 + 32x^6 + 55$
(**) $x^8 + x + 17$
(**) $x^9 + 2x + 10$
$x^{10} + x^2 + x + 10$
(**) $x^{11} + 12x + 10$
(**) $x^{12} + x + 44$
(**) $x^{13} + 3x + 2$
(**) $x^{14} + x + 43$
(**) $x^{15} + 2x + 6$
(**) $x^{16} + x + 2$
(**) $x^{17} + 2x + 30$
$x^{18} + x^2 + 5x + 44$
(**) $x^{19} + 2x + 2$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **61**

$x^{20} + x^2 + x + 2$
(**) $x^{21} + 4x + 10$
(**) $x^{22} + 2x + 2$
(**) $x^{23} + x + 26$
$x^{24} + x^2 + x + 30$
(**) $x^{25} + 13x + 2$
$x^{26} + x^2 + x + 10$
$x^{27} + x^2 + x + 6$
$x^{28} + x^2 + 2x + 7$
(**) $x^{29} + 4x + 2$
$x^{30} + x^2 + x + 10$
(**) $x^{31} + 11x + 2$
(**) $x^{32} + x + 17$
(**) $x^{33} + 2x + 6$
(**) $x^{35} + 2x + 6$
$x^{36} + x^2 + x + 55$
(**) $x^{37} + 5x + 2$
$x^{41} + x^2 + x + 51$
(**) $x^{42} + x + 31$
(**) $x^{44} + x + 31$
$x^{53} + x^2 + x + 43$
(**) $x^{55} + 2x + 7$
$x^{56} + x^2 + x + 10$
(**) $x^{59} + 24x + 2$
$x^{66} + x^2 + 2x + 7$

Polinomios primitivos para el primo **67**

(**) $x^2 + 60x + 18$
(*) $x^3 + 21x^2 + 36$
(**) $x^4 + 39x + 28$
(*) $x^5 + 53x^2 + 16$
$x^6 + 32x^5 + 3x^4 + 30x^3 + 39x^2 + 10x + 48$
(**) $x^7 + 9x + 39$

Polinomios primitivos para el primo **71**

(**) $x^2 + 35x + 33$
(*) $x^3 + 63x^2 + 29$
(*) $x^4 + 67x^3 + 53$
(**) $x^5 + 13x + 50$
(**) $x^6 + 69x + 31$
(*) $x^7 + 26x^4 + 58$

Polinomios primitivos para el primo **73**

(**) $x^2 + 40x + 28$
(**) $x^3 + 58x + 29$
(**) $x^4 + x + 20$
(*) $x^5 + 9x^3 + 40$
(*) $x^6 + 19x^5 + 39$
(*) $x^7 + 50x^2 + 39$

Polinomios primitivos para el primo **79**

(**) $x^2 + 2x + 47$
(*) $x^3 + 49x^2 + 76$
(*) $x^4 + 67x^3 + 34$
(**) $x^5 + 78x + 42$
(*) $x^6 + 50x^5 + 53$

Polinomios primitivos para el primo **79**

(*) $x^7 + 32x^4 + 5$

Polinomios primitivos para el primo **83**

(**) $x^2 + x + 19$
(**) $x^3 + 34x + 33$
(**) $x^4 + 37x + 15$
(**) $x^5 + 51x + 63$
(*) $x^6 + 30x^5 + 5$
(*) $x^7 + 17x^2 + 64$

Polinomios primitivos para el primo **89**

(**) $x^2 + 43x + 30$
(**) $x^3 + 29x + 24$
(*) $x^4 + 55x^3 + 19$
(*) $x^5 + 65x^2 + 35$
(**) $x^6 + 75x + 83$
(*) $x^7 + 67x^6 + 61$

Polinomios primitivos para el primo **97**

(**) $x^2 + 74x + 80$
(*) $x^3 + 52x^2 + 38$
(*) $x^4 + 23x^3 + 5$
(**) $x^5 + 83x + 76$
(*) $x^6 + 11x^5 + 84$
(**) $x^7 + 71x + 83$

Polinomios primitivos para el primo **101**

(**) $x^2 + 45x + 59$
(*) $x^3 + 30x^2 + 63$
(**) $x^4 + 98x + 75$
(**) $x^5 + 63x + 55$
(*) $x^6 + 63x^5 + 90$
(*) $x^7 + 53x^3 + 26$

Polinomios primitivos para el primo **103**

(**) $x^2 + 57x + 48$
(*) $x^3 + 19x^2 + 49$
(**) $x^4 + 61x + 85$
(**) $x^5 + 22x + 55$
(**) $x^6 + 96x + 67$
(*) $x^7 + 17x^3 + 83$

Polinomios primitivos para el primo **107**

(**) $x^2 + 78x + 98$
(*) $x^3 + 56x^2 + 29$
(**) $x^4 + 39x + 78$
(*) $x^5 + 36x^3 + 52$
(*) $x^6 + 7x^5 + 73$
(**) $x^7 + 83x + 64$

Polinomios primitivos para el primo **109**

(**) $x^2 + 24x + 18$
(*) $x^3 + 52x^2 + 95$
(*) $x^4 + 57x^3 + 58$
(**) $x^5 + 45x + 95$
(*) $x^6 + 62x^5 + 95$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **109**

(*) $x^7 + 44x^5 + 47$

Polinomios primitivos para el primo **113**

(**) $x^2 + 36x + 33$
(*) $x^3 + 11x^2 + 59$
(*) $x^4 + 94x^3 + 45$
(**) $x^5 + 82x + 80$
(*) $x^6 + 84x^5 + 12$
(*) $x^7 + 22x^2 + 21$

Polinomios primitivos para el primo **127**

(**) $x^2 + 13x + 116$
(**) $x^2 + x + 3$
(*) $x^3 + 125x^2 + 13$
(**) $x^3 + x + 15$
(**) $x^4 + 42x + 56$
(**) $x^4 + x + 3$
(*) $x^5 + 99x^3 + 113$
(**) $x^5 + x + 11$
(*) $x^6 + 86x^5 + 48$
(**) $x^6 + x + 29$
(*) $x^7 + 40x^3 + 62$
(**) $x^7 + x + 26$
(**) $x^8 + x + 55$
(**) $x^9 + x + 17$
(**) $x^{10} + 3x + 6$
(**) $x^{11} + x + 62$
(**) $x^{12} + x + 48$
(**) $x^{13} + x + 11$
(**) $x^{14} + x + 29$
(**) $x^{15} + x + 13$
(**) $x^{16} + x + 3$
(**) $x^{17} + x + 71$
$x^{18} + x^2 + x + 57$
(**) $x^{20} + x + 58$
(**) $x^{21} + x + 15$
(**) $x^{22} + 14x + 29$
(**) $x^{23} + x + 13$
(**) $x^{24} + x + 116$
(**) $x^{25} + 29x + 18$
(**) $x^{26} + x + 112$
(**) $x^{27} + x + 88$
(**) $x^{28} + 7x + 23$
(**) $x^{29} + 4x + 11$
(**) $x^{30} + 2x + 106$
(**) $x^{31} + 13x + 36$
(**) $x^{32} + x + 93$
(**) $x^{34} + x + 3$
(**) $x^{36} + x + 55$
(**) $x^{37} + 19x + 11$
(**) $x^{41} + x + 62$
(**) $x^{48} + x + 43$
(**) $x^{54} + x + 91$
(**) $x^{59} + 3x + 15$
(**) $x^{67} + 5x + 60$

Polinomios primitivos para el primo **131**

(**) $x^2 + 47x + 72$
(*) $x^3 + 5x^2 + 25$
(**) $x^4 + 48x + 23$
(*) $x^5 + 99x^2 + 100$
(*) $x^6 + 33x^5 + 2$
(*) $x^7 + 115x^2 + 81$

Polinomios primitivos para el primo **137**

(**) $x^2 + 72x + 21$
(**) $x^3 + 128x + 43$
(*) $x^4 + 67x^3 + 57$
(*) $x^5 + 80x^4 + 108$
(*) $x^6 + 22x^5 + 33$
(*) $x^7 + 31x^4 + 92$

Polinomios primitivos para el primo **139**

(**) $x^2 + 105x + 88$
(**) $x^3 + 126x + 37$
(**) $x^4 + 24x + 50$
(*) $x^5 + 50x^3 + 35$
(**) $x^6 + 8x + 104$
(*) $x^7 + 123x^5 + 122$

Polinomios primitivos para el primo **149**

(**) $x^2 + 13x + 43$
(**) $x^3 + 22x + 2$
(**) $x^4 + 118x + 66$
(*) $x^5 + 8x^2 + 77$
(*) $x^6 + 74x^5 + 111$
(*) $x^7 + 20x^3 + 126$

Polinomios primitivos para el primo **151**

(**) $x^2 + 74x + 7$
(**) $x^3 + 64x + 40$
(**) $x^4 + 2x + 146$
(**) $x^5 + 122x + 18$
(**) $x^6 + 91x + 89$
(*) $x^7 + 23x^5 + 36$

Polinomios primitivos para el primo **157**

(**) $x^2 + 63x + 96$
(**) $x^3 + 38x + 55$
(*) $x^4 + 126x^3 + 55$
(*) $x^5 + 131x^2 + 38$
(**) $x^6 + 145x + 66$
(*) $x^7 + 137x^5 + 133$

Polinomios primitivos para el primo **163**

(**) $x^2 + 87x + 92$
(*) $x^3 + 111x^2 + 62$
(*) $x^4 + 81x^3 + 159$
(*) $x^5 + x^3 + 33$
(*) $x^6 + 67x^5 + 45$
(*) $x^7 + 87x^6 + 35$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **167**
(**) $x^2 + 33x + 45$
(**) $x^3 + 113x + 7$
(**) $x^4 + 109x + 40$
(**) $x^5 + 33x + 124$
(*) $x^6 + x^5 + 155$
(**) $x^7 + 85x + 42$

Polinomios primitivos para el primo **173**
(**) $x^2 + 62x + 70$
(**) $x^3 + 171x + 70$
(**) $x^4 + 17x + 71$
(*) $x^5 + 143x^4 + 97$
(*) $x^6 + 126x^5 + 19$

Polinomios primitivos para el primo **179**
(**) $x^2 + 5x + 98$
(**) $x^3 + 90x + 15$
(**) $x^4 + 44x + 162$
(*) $x^5 + 16x^2 + 173$
(**) $x^6 + 96x + 175$
(*) $x^7 + 156x^3 + 110$

Polinomios primitivos para el primo **181**
(**) $x^2 + 168x + 47$
(**) $x^3 + 176x + 76$
(*) $x^4 + 100x^3 + 158$
(*) $x^5 + 171x^4 + 23$
(**) $x^6 + 11x + 97$
(*) $x^7 + 143x^3 + 85$

Polinomios primitivos para el primo **191**
(**) $x^2 + 131x + 62$
(**) $x^3 + 9x + 120$
(**) $x^4 + 35x + 168$
(*) $x^5 + 34x^3 + 59$
(**) $x^6 + 78x + 143$
(*) $x^7 + 166x^6 + 79$

Polinomios primitivos para el primo **193**
(**) $x^2 + 170x + 152$
(**) $x^3 + 170x + 103$
(**) $x^4 + 124x + 22$
(*) $x^5 + 20x^3 + 116$
(*) $x^6 + 38x^5 + 58$
(*) $x^7 + 36x^5 + 41$

Polinomios primitivos para el primo **197**
(**) $x^2 + 183x + 162$
(**) $x^3 + 147x + 95$
(*) $x^4 + 111x^3 + 67$
(*) $x^5 + 39x^3 + 32$
(**) $x^6 + 66x + 166$
(*) $x^7 + 144x^2 + 45$

Polinomios primitivos para el primo **199**
(**) $x^2 + 183x + 190$

Polinomios primitivos para el primo **199**
(**) $x^3 + 130x + 115$
(**) $x^4 + 96x + 108$
(**) $x^5 + 129x + 50$
(**) $x^6 + 49x + 192$
(*) $x^7 + 198x^4 + 128$

Polinomios primitivos para el primo **211**
(**) $x^2 + 197x + 85$
(**) $x^3 + 140x + 44$
(**) $x^4 + 41x + 141$
(*) $x^5 + 139x^4 + 45$
(**) $x^6 + 88x + 57$
(*) $x^7 + 58x^5 + 81$

Polinomios primitivos para el primo **223**
(**) $x^2 + 34x + 67$
(**) $x^3 + 52x + 177$
(**) $x^4 + x + 173$
(*) $x^5 + 19x^2 + 218$
(**) $x^6 + 105x + 45$
(*) $x^7 + 181x^6 + 211$

Polinomios primitivos para el primo **227**
(**) $x^2 + 68x + 98$
(**) $x^3 + 53x + 97$
(*) $x^4 + 138x^3 + 115$
(*) $x^5 + 93x^2 + 121$
(**) $x^6 + 220x + 168$
(*) $x^7 + 188x^3 + 175$

Polinomios primitivos para el primo **229**
(**) $x^2 + 184x + 31$
(*) $x^3 + 6x^2 + 191$
(*) $x^4 + 59x^3 + 179$
(*) $x^5 + 226x^2 + 152$
(*) $x^6 + 131x^5 + 28$
(*) $x^7 + 122x^6 + 6$

Polinomios primitivos para el primo **233**
(**) $x^2 + 17x + 212$
(*) $x^3 + 86x^2 + 151$
(**) $x^4 + 30x + 43$
(*) $x^5 + 219x^2 + 186$
(*) $x^6 + 172x^5 + 190$
(**) $x^7 + 60x + 20$

Polinomios primitivos para el primo **239**
(**) $x^2 + 233x + 179$
(*) $x^3 + 33x^2 + 4$
(**) $x^4 + 66x + 69$
(*) $x^5 + 164x^3 + 113$
(*) $x^6 + 137x^5 + 65$
(**) $x^7 + 7x + 155$

Polinomios primitivos para el primo **241**
(**) $x^2 + 210x + 31$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **241**
(*) $x^3 + 178x^2 + 234$
(*) $x^4 + 73x^3 + 132$
(*) $x^5 + 32x^4 + 142$
(*) $x^6 + 205x^5 + 112$
(*) $x^7 + 124x^2 + 199$

Polinomios primitivos para el primo **251**
(**) $x^2 + 204x + 178$
(**) $x^3 + 133x + 112$
(*) $x^4 + 28x^3 + 185$
(*) $x^5 + 175x^3 + 190$
(**) $x^6 + 140x + 220$
(*) $x^7 + 183x^5 + 52$

Polinomios primitivos para el primo **257**
(**) $x^2 + 186x + 115$
(**) $x^2 + x + 5$
(*) $x^3 + 149x^2 + 86$
(**) $x^3 + x + 5$
(*) $x^4 + 240x^3 + 218$
(**) $x^4 + x + 3$
(*) $x^5 + 112x^3 + 10$
(**) $x^5 + x + 10$
(**) $x^6 + x + 41$
(*) $x^7 + 25x^5 + 132$
(**) $x^7 + x + 6$
(**) $x^8 + x + 19$
(**) $x^9 + x + 6$
(**) $x^{10} + x + 39$
(**) $x^{11} + x + 14$
(**) $x^{12} + x + 3$
(**) $x^{13} + x + 19$
(**) $x^{14} + x + 93$
(**) $x^{15} + x + 43$
(**) $x^{16} + x + 105$
(**) $x^{17} + 2x + 19$
(**) $x^{18} + x + 86$
(**) $x^{19} + x + 7$
(**) $x^{20} + x + 47$
(**) $x^{21} + x + 7$
(**) $x^{22} + x + 54$
(**) $x^{23} + x + 41$
(**) $x^{24} + x + 106$
(**) $x^{26} + x + 12$
(**) $x^{28} + x + 212$
(**) $x^{30} + x + 160$
(**) $x^{32} + x + 10$
(**) $x^{35} + x + 14$
(**) $x^{41} + x + 6$
(**) $x^{43} + x + 5$
(**) $x^{45} + x + 53$
(**) $x^{48} + x + 77$
(**) $x^{59} + x + 85$

Polinomios primitivos para el primo **263**
(**) $x^2 + 110x + 224$

Polinomios primitivos para el primo **263**
(*) $x^3 + 49x^2 + 74$
(*) $x^4 + 221x^3 + 194$
(*) $x^5 + 228x^4 + 39$
(*) $x^6 + 235x^5 + 215$
(*) $x^7 + 239x^6 + 62$

Polinomios primitivos para el primo **269**
(**) $x^2 + 125x + 76$
(*) $x^3 + 223x^2 + 107$
(**) $x^4 + 194x + 250$
(**) $x^5 + 162x + 75$
(**) $x^6 + 169x + 259$
(*) $x^7 + 100x^2 + 109$

Polinomios primitivos para el primo **271**
(**) $x^2 + 150x + 182$
(**) $x^3 + 75x + 180$
(*) $x^4 + 3x^3 + 95$
(*) $x^5 + 127x^4 + 112$
(*) $x^6 + 74x^5 + 161$

Polinomios primitivos para el primo **277**
(**) $x^2 + 139x + 178$
(**) $x^3 + 75x + 259$
(**) $x^4 + 139x + 205$
(*) $x^5 + 242x^2 + 221$
(**) $x^6 + 175x + 43$
(*) $x^7 + 218x^5 + 197$

Polinomios primitivos para el primo **281**
(**) $x^2 + 83x + 26$
(**) $x^3 + 201x + 22$
(*) $x^4 + 141x^3 + 23$
(*) $x^5 + 200x^4 + 278$
(**) $x^6 + 117x + 227$
(*) $x^7 + 248x^6 + 174$

Polinomios primitivos para el primo **283**
(**) $x^2 + 181x + 153$
(*) $x^3 + 261x^2 + 99$
(*) $x^4 + 28x^3 + 220$
(*) $x^5 + 123x^4 + 117$
(*) $x^6 + 277x^5 + 249$
(*) $x^7 + 137x^4 + 129$

Polinomios primitivos para el primo **293**
(**) $x^2 + 43x + 101$
(**) $x^3 + 36x + 98$
(*) $x^4 + 215x^3 + 219$
(**) $x^5 + 219x + 208$
(**) $x^6 + 241x + 168$

Polinomios primitivos para el primo **307**
(**) $x^2 + 288x + 186$
(**) $x^3 + 224x + 87$
(**) $x^4 + 38x + 161$

Universitat d'Alacant
Universidad de Alicante

Polinomios primitivos para el primo **307**

(*) $x^5 + 55x^3 + 177$
(**) $x^6 + 124x + 106$
(*) $x^7 + 154x^3 + 39$

Polinomios primitivos para el primo **311**

(**) $x^2 + 279x + 205$
(*) $x^3 + 227x^2 + 267$
(*) $x^4 + 144x^3 + 230$
(*) $x^5 + 155x^3 + 160$
(**) $x^6 + 305x + 22$
(**) $x^7 + 130x + 235$

Polinomios primitivos para el primo **313**

(**) $x^2 + 187x + 45$
(**) $x^3 + 64x + 31$
(**) $x^4 + 71x + 167$
(*) $x^5 + 233x^4 + 110$
(*) $x^6 + 256x^5 + 34$
(**) $x^7 + 50x + 130$

Polinomios primitivos para el primo **509**

(**) $x^2 + x + 2$
(**) $x^3 + x + 3$
(**) $x^4 + x + 22$
(**) $x^5 + x + 12$
(**) $x^6 + x + 13$
(**) $x^7 + x + 15$
(**) $x^8 + x + 46$
(**) $x^9 + x + 7$
(**) $x^{10} + x + 3$
(**) $x^{11} + x + 22$
(**) $x^{12} + x + 149$
(**) $x^{13} + x + 119$
(**) $x^{14} + x + 34$
(**) $x^{15} + x + 41$
(**) $x^{16} + x + 31$
(**) $x^{17} + x + 48$
(**) $x^{19} + x + 31$
(**) $x^{21} + x + 48$
(**) $x^{24} + x + 132$
(**) $x^{25} + x + 58$
(**) $x^{28} + x + 112$
(**) $x^{29} + x + 77$
(**) $x^{35} + x + 161$
(**) $x^{49} + x + 7$

Polinomios primitivos para el primo **1021**

(**) $x^2 + x + 10$
(**) $x^3 + x + 34$
(**) $x^4 + x + 22$
(**) $x^5 + x + 43$
(**) $x^6 + x + 31$
(**) $x^7 + x + 10$
(**) $x^8 + x + 22$
(**) $x^9 + x + 31$
(**) $x^{10} + x + 22$

Polinomios primitivos para el primo **1021**

(**) $x^{11} + x + 66$
(**) $x^{12} + x + 53$
(**) $x^{13} + x + 82$
(**) $x^{14} + x + 31$
(**) $x^{15} + x + 34$
(**) $x^{16} + x + 22$
(**) $x^{18} + x + 31$
(**) $x^{20} + x + 127$
(**) $x^{21} + x + 143$
(**) $x^{24} + x + 175$
(**) $x^{25} + x + 37$
(**) $x^{28} + x + 46$

Polinomios primitivos para el primo **2053**

(**) $x^2 + x + 20$
(**) $x^3 + x + 24$
(**) $x^4 + x + 5$
(**) $x^5 + x + 24$
(**) $x^6 + x + 209$
(**) $x^7 + x + 50$
(**) $x^8 + x + 43$
(**) $x^9 + x + 32$
(**) $x^{10} + x + 6$
(**) $x^{11} + x + 33$
(**) $x^{12} + x + 111$
(**) $x^{13} + x + 6$
(**) $x^{14} + x + 35$
(**) $x^{15} + x + 38$
(**) $x^{16} + x + 102$
(**) $x^{18} + x + 32$
(**) $x^{19} + x + 35$
(**) $x^{20} + 2x + 6$
(**) $x^{21} + x + 5$
(**) $x^{28} + x + 162$
(**) $x^{37} + x + 33$

Polinomios primitivos para el primo **3187**

(**) $x^2 + x + 5$
(**) $x^3 + x + 25$
(**) $x^4 + x + 5$
(**) $x^5 + x + 44$
(**) $x^6 + x + 23$
(**) $x^7 + x + 25$
(**) $x^8 + x + 20$
(**) $x^9 + x + 36$
(**) $x^{10} + x + 13$
(**) $x^{11} + x + 16$
(**) $x^{12} + x + 75$
(**) $x^{14} + x + 42$
(**) $x^{15} + x + 66$
(**) $x^{16} + x + 108$
(**) $x^{17} + x + 60$
(**) $x^{18} + x + 2$
(**) $x^{20} + x + 120$
(**) $x^{21} + x + 211$
(**) $x^{23} + x + 66$

Universitat d'Alicant
Universidad de Alicante

Polinomios primitivos para el primo **3187**

(**) $x^{24} + x + 236$
(**) $x^{25} + x + 35$
(**) $x^{29} + x + 172$
(**) $x^{37} + x + 44$
(**) $x^{43} + x + 294$

Polinomios primitivos para el primo **65537**

(**) $x^{2} + x + 3$
(**) $x^{3} + x + 7$
(**) $x^{4} + x + 6$
(**) $x^{5} + x + 3$

Polinomios primitivos para el primo **65537**

(**) $x^{6} + x + 14$
(**) $x^{7} + x + 3$
(**) $x^{8} + x + 20$
(**) $x^{9} + x + 10$
(**) $x^{10} + x + 6$
(**) $x^{11} + x + 27$
(**) $x^{12} + x + 11$
(**) $x^{13} + x + 56$
(**) $x^{14} + x + 108$
(**) $x^{15} + x + 20$
(**) $x^{19} + x + 47$

Primo	Raíz primitiva
2	1
3	2
5	2
7	3
11	2
13	2
17	3
19	2
23	5
29	2
31	3
37	2
41	6
43	3
47	5
53	2
59	2
61	2
67	2
71	7
73	5
79	3
83	2
89	3

Primo	Raíz primitiva
97	5
101	2
103	5
107	2
109	6
113	3
127	3
131	2
137	3
139	2
149	2
151	6
157	5
163	2
167	5
173	2
179	2
181	2
191	19
193	5
197	2
199	3
211	2
223	3

Primo	Raíz primitiva
227	2
229	6
233	3
239	7
241	7
251	6
257	3
263	5
269	2
271	6
277	5
281	3
283	3
293	2
307	5
311	17
313	10
509	2
1021	10
2053	2
3187	2
65537	3

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

El nunca me lo exigió,
pero, tras su falta, sentí que se lo debía.
Sirva de estímulo a las siguientes generaciones.