ESTUDIO DE LAS CADENAS PSEUDO ALEATORIAS GENERADAS POR LOS AUTÓMATAS CELULARES UNIDIMENSIONALES DE DOS ESTADOS, IMPLEMENTACIÓN Y APLICACIONES A CRIPTOGRAFÍA.

LUIS DANIEL GONZÁLEZ OGLIASTRI
FEDERICO JOSÉ CHAVES NIÑO

UNIVERSIDAD INDUSTRIAL DE SANTANDER UIS
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2006

ESTUDIO DE LAS CADENAS PSEUDO ALEATORIAS GENERADAS POR LOS AUTÓMATAS CELULARES UNIDIMENSIONALES DE DOS ESTADOS, IMPLEMENTACIÓN Y APLICACIONES A CRIPTOGRAFÍA.

LUIS DANIEL GONZÁLEZ OGLIASTRI
FEDERICO JOSÉ CHAVES NIÑO

Trabajo de grado para optar por el título de Ingeniero de Sistemas

Director:
Rafael Fernando Isaacs Giraldo

Codirector:
Fernando Ruiz Díaz

UNIVERSIDAD INDUSTRIAL DE SANTANDER UIS
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2006

DEDICATORIA

A mi mamá por haberme sacado adelante y por haber hecho que nunca me faltara nada.

A mi abuela , a mi papá y a mis hermanos por su apoyo incondicional.

A Ivette una persona difícil de encontrar, fácil de querer e imposible de olvidar.

A mis amigos por acompañarme en todos los momentos.

A Luis Daniel por haberme soportado todo este tiempo.

Federico

DEDICATORIA

A mis padres por todos sus sacrificios.

A toda mi familia.

A Carolyn porque sin ella nada sería igual.

A Federico por soportarme 14 años.

Luis Daniel

AGRADECIMIENTOS

A nuestro director de proyecto, Rafael Isaacs por su constante colaboración y apoyo.

A nuestros compañeros por todo el tiempo compartido durante la carrera.

A todas las personas que influyeron en nuestro paso por la universidad.

A la Universidad Industrial de Santander.

CONTENIDO

INTRODUCCIÓN **15**

1. PRESENTACIÓN DEL PROYECTO **17**
1.1. PRESENTACIÓN DEL INFORME 17
1.2. DESCRIPCIÓN DEL PROYECTO 17
1.2.1. Objetivo General 17
1.2.2. Objetivos Específicos 17
1.3. JUSTIFICACIÓN 18

2. MARCO TEÓRICO **20**
2.1. CRIPTOLOGÍA 20
2.1.1. Criptografía 20
2.1.2. Criptoanálisis 35
2.2. generadorES DE BITS PSEUDO ALEATORIOS 37
2.2.1. Ejemplos de generadores de *bits* 39
2.3. AUTÓMATAS CELULARES 41
2.3.1. Autómatas celulares de Wolfram 43
2.3.2. Juego de la vida 45
2.4. AUTÓMATAS CELULARES COMO generadorES DE BITS PSEUDO ALEATORIOS 48
2.5. LAS PRUEBAS ESTADÍSTICAS 49
2.5.1. Los Postulados de Golomb 49
2.5.2. Prueba de Frecuencias (*Monobit test*) 51
2.5.3. Prueba de Series (*Two-bit test*) 51
2.5.4. Prueba del Poker 51
2.5.5. Prueba de rachas 52
2.5.6. Prueba de Autocorrelación 52
2.5.7. Prueba estadística universal de Maurer 53
2.5.8. Prueba Chi-cuadrado 53

3. PROPUESTA **56**
3.1. DESARROLLO DE LOS generadorES 56
3.1.1. Metodología de las pruebas 57
3.1.2. Método 1 60
3.1.3. Método 2 63
3.1.4. Incremento en la velocidad de generación de la sucesión cifrante 67
3.1.5. Método 3 69
3.2. CRIPTOANÁLISIS 71
3.2.1. Claves Débiles 71

3.2.2. Fuerza Bruta 72
3.2.3. Inversión de un Autómata Celular 73

4. DESARROLLO DE LA APLICACIÓN **74**
4.1. LA CLASE AUTÓMATA 74
4.2. IMPLEMENTACIÓN DE LOS MÉTODOS 80
4.3. ENTORNO VISUAL 82

5. CONCLUSIONES **89**

6. RECOMENDACIONES **90**

ANEXO A **91**

BIBLIOGRAFÍA **104**

Índice de figuras

1. Proceso general cifrado/descifrado. 21
2. Esquema General de Cifrado en Flujo 24
3. Red de Feistel 28
4. Esquema de la funcion f del algoritmo DES 29
5. Cálculo de las K para el algoritmo DES. 30
6. Transmisión de la información usando algoritmos asimétricos 31
7. Autenticación de información empleando algoritmos asimétricos. 31
8. Generación de *bits* Pseudo Aleatorios 38
9. Arreglo de 10 Células 44
10. Vecindad de Tamaño 3 44
11. Vecindad de Tamaño 4 45
12. Regla 105 45
13. Evolución de la regla 105 45
14. Frontera periódica 46
15. Enumeración de las reglas 46
16. Evolución de un deslizador 47
17. Cañon de deslizadores 48
18. Evolución de la regla 30 56
19. Método 1. Paso 1. 61
20. Propagación del cambio de un *bit* 62
21. Selección Aleatoria de Células 64
22. Celdas elegidas del generador 67
23. Representación de la Regla 21910 71
24. Evolución de la regla 21910 con una semilla aleatoria 71
25. Evolución de una clave débil en la regla 21910 72
26. Funcionamiento del procedimiento evolucionar 76
27. Función Devolverunbyte 78
28. Cabecera de los archivos encriptados 81
29. Diálogo Principal 82
30. Cifrar Archivo 84
31. Diálogo Encriptar Archivo 84
32. Selección del archivo a encriptar 85
33. Archivo Seleccionado para Encriptar 85
34. Ejecución del Encriptado 86
35. Cancelación del Proceso 86
36. Proceso Terminado 87
37. Diálogo para escribir el texto a encriptar 87
38. Diálogo Encriptar Texto 88
39. Diálogo Desencriptar 88

Índice de cuadros

1. Autómatas que pasaron las pruebas estadísticas 60
2. Autómatas que pasaron la prueba Chi Cuadrado 65
3. Autómatas que pasaron la prueba Chi Cuadrado con el algoritmo mejorado 69
4. Resultados Prueba de Maurer con clave de 256 bits 70
5. Resultados Prueba de Maurer con clave de 512 bits 70
6. Resultados Prueba de Maurer con clave de 1024 bits 70
7. Autómatas que pasaron la prueba con el generador 11730 91
8. Autómatas que pasaron la prueba con el generador 15420 91
9. Autómatas que pasaron la prueba con el generador 21846 91
10. Autómatas que pasaron la prueba con el generador 21913 91
11. Autómatas que pasaron la prueba con el generador 21925 92
12. Autómatas que pasaron la prueba con el generador 22105 92
13. Autómatas que pasaron la prueba con el generador 22118 92
14. Autómatas que pasaron la prueba con el generador 22122 92
15. Autómatas que pasaron la prueba con el generador 22166 92
16. Autómatas que pasaron la prueba con el generador 22185 93
17. Autómatas que pasaron la prueba con el generador 22937 93
18. Autómatas que pasaron la prueba con el generador 22953 93
19. Autómatas que pasaron la prueba con el generador 23126 93
20. Autómatas que pasaron la prueba con el generador 23129 94
21. Autómatas que pasaron la prueba con el generador 23141 94
22. Autómatas que pasaron la prueba con el generador 23142 94
23. Autómatas que pasaron la prueba con el generador 23146 94
24. Autómatas que pasaron la prueba con el generador 25941 94
25. Autómatas que pasaron la prueba con el generador 25962 95
26. Autómatas que pasaron la prueba con el generador 26198 95
27. Autómatas que pasaron la prueba con el generador 26966 95
28. Autómatas que pasaron la prueba con el generador 27221 95
29. Autómatas que pasaron la prueba con el generador 27225 96
30. Autómatas que pasaron la prueba con el generador 27226 96
31. Autómatas que pasaron la prueba con el generador 27242 96
32. Autómatas que pasaron la prueba con el generador 27290 96
33. Autómatas que pasaron la prueba con el generador 27305 96
34. Autómatas que pasaron la prueba con el generador 34425 97
35. Autómatas que pasaron la prueba con el generador 38245 97
36. Autómatas que pasaron la prueba con el generador 38249 97
37. Autómatas que pasaron la prueba con el generador 38293 97
38. Autómatas que pasaron la prueba con el generador 38297 97
39. Autómatas que pasaron la prueba con el generador 38310 98

40. Autómatas que pasaron la prueba con el generador 38489 98
41. Autómatas que pasaron la prueba con el generador 38501 98
42. Autómatas que pasaron la prueba con el generador 38506 98
43. Autómatas que pasaron la prueba con el generador 38550 98
44. Autómatas que pasaron la prueba con el generador 38565 99
45. Autómatas que pasaron la prueba con el generador 39253 99
46. Autómatas que pasaron la prueba con el generador 39257 99
47. Autómatas que pasaron la prueba con el generador 39274 99
48. Autómatas que pasaron la prueba con el generador 39317 99
49. Autómatas que pasaron la prueba con el generador 39510 100
50. Autómatas que pasaron la prueba con el generador 39513 100
51. Autómatas que pasaron la prueba con el generador 39573 100
52. Autómatas que pasaron la prueba con el generador 39574 100
53. Autómatas que pasaron la prueba con el generador 39593 101
54. Autómatas que pasaron la prueba con el generador 42326 101
55. Autómatas que pasaron la prueba con el generador 42329 101
56. Autómatas que pasaron la prueba con el generador 42393 101
57. Autómatas que pasaron la prueba con el generador 42394 101
58. Autómatas que pasaron la prueba con el generador 42406 102
59. Autómatas que pasaron la prueba con el generador 42409 102
60. Autómatas que pasaron la prueba con el generador 42585 102
61. Autómatas que pasaron la prueba con el generador 42646 102
62. Autómatas que pasaron la prueba con el generador 43353 102
63. Autómatas que pasaron la prueba con el generador 43369 103
64. Autómatas que pasaron la prueba con el generador 43370 103
65. Autómatas que pasaron la prueba con el generador 57630 103

TÍTULO

ESTUDIO DE LAS CADENAS PSEUDO ALEATORIAS GENERADAS POR LOS AUTÓMATAS CELULARES UNIDIMENSIONALES DE DOS ESTADOS, IMPLEMENTACIÓN Y APLICACIONES A CRIPTOGRAFÍA.[1]

AUTORES

CHAVES NIÑO FEDERICO JOSÉ, GONZÁLEZ OGLIASTRI LUIS DANIEL.[2]

PALABRAS CLAVE

Autómatas celulares, criptografía, cifrado en flujo, sucesiones pseudo aleatorioas, Wolfram, pruebas estadísticas.

DESCRIPCIÓN

Los autómatas celulares fueron propuestos por Jhon von Newman en el anõ de 1936. Él tenia la idea de crear una máquina capaz de autoreproducirse y con la ayuda de Stanislav Ulam pudo lograr su objetivo. Más adelante Jhon Conway creo "El Juego de la Vida", tal vez el autómata celular mas famoso. Algunas configuraciones de este autómata tienen comportamientos tan complejos como generar números primos. Hacia los anõs 80, Stephen Wolfram presenta su estudio sobre autómatas celulares lineales. En 1986 el mismo Wolfram propone la utilización de sus autómatas en la criptografía como generadores de sucesiones de bits pseudo aleatorias.

Por el lado de la criptografía el proyecto se centra en los sistemas de cifrado en flujo, estos derivan del método propuesto por Vernam, con la diferencia que la sucesión cifrante no es aleatoria sino que se genera pseudo aleatoriamente a partir de una semilla.

Se presentan tres métodos de generación de suscesiones pseudo aleatorias de bits. El primero es una variación al propuesto por Wolfram con la diferencia que se usan vecindades de tamaño 4 en vez de 3. Este método es inseguro por cuando no presenta ningun obstaculo para evitar el criptoanálisis por clave reusada.

El segundo método elimina esta debilidad añadiendo un generador de números pseudo aleatorios de tal manera que no se elige siempre la misma célula sino que permite la elección de diferentes celulas que dependen de una segunda clave llamada "nonce". Sin embargo este segundo método es muy lento debido a que se necesitan muchas evoluciones de los autómatas para la generación de un solo bit de sucesión cifrante.

Finalmente con el método 3 se solucionan los problemas de velocidad presentados el el método anterior mediante la selección de múltiples columnas para el autómata del generador de números pseudoaleatorios.

[1] Trabajo de Grado

[2] Facultad de Ingenierías Físico - Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Universidad Industrial de Santander. Directores: Doctor Rafael Isaacs Giraldo, Magíster Fernando Ruíz Díaz.

TITLE

STUDY ABOUT PSEUDO RANDOM SEQUENCES GENERATED BY UNIDIMENTIONAL TWO-STATE CELLULAR AUTOMATA, IMPLEMENTATION AND USES IN CRIPTOGRAPHY.[3]

AUTHORS

CHAVES NIÑO FEDERICO JOSÉ, GONZÁLEZ OGLIASTRI LUIS DANIEL.[4]

KEY WORDS

Cellular automata, criptography, stream cipher, pseudo random bit sequences, Wolfram, statistical tests.

ABSTRACT

Cellular automata where proposed by Jhon von Newman around 1936. He had the idea of creating a self-reproductive machine and, along with his friend Stanislav Ulam could acomplish his goal. The next big step in cellular automata development was the creation of Jhon Conway's "Game of Life" wich can be considered as the most famous one. Some configurations of this automaton have such complex behaviours as generating prime numbers. In the 80's, Stephen Wolfram presented his studies about linear cellular automata. In the year 1986, Wolfram proposed the utilization of cellular automata in criptography as pseudo random bits sequences generators.

About criptography, the proyect is focused mostly in stream ciphers, theese are approaches to the cipher proposed by Vernam, with the difference that, instead of using a random keystream as long as the plaintext, it uses a pseudo random keystream generated by an criptographically secure algorithm.

In the book appear three different methods to generate pseudo random bits sequences. The first one is a variation of the one proposed by Wolfram but instead of using 3 neighbors it uses 4. This is an insecure method because it does not implement any obstacle to avoid reused key criptanalysis.

The second method fixes this problem by adding a pseudo random number generador used to choose differen cells in every evolution of the automaton, that depends on a second key called "nonce". However this method is very slow due to it needs to many evolution of the automata in order to generate one single bit of the keystream.

Finally with the third method all speed problems present in the previous method are solved by choosing multiples cells in the automaton used in the pseudo random number generator.

[3]Thesis

[4]Faculty of Physical - Mechanical Engineerings. Department of Systems Engineering. Universidad Industrial de Santander. Tutors: Doctor Rafael Isaacs Giraldo, Master Fernando Ruíz Díaz.

INTRODUCCIÓN

Desde los principios de la escritura se han venido desarrollando formas de transmitir secretos. En la antigüedad los griegos utilizaron un cilindro llamado *scytale* alrededor del cual se enrrollaba una tira de cuero. Al escribir un mensaje sobre el cuero y desenrollarlo se veía una lista de letras sin sentido la cual se enviaba a su destinatario que podía leerlo pues poseía un cilindro del mismo diámetro. Más adelante en la historia, Julio Cesar, el emperador romano, empleó un sistema de cifrado que consistía en sustituir la letra a encriptar por otra que se encontraba tres posiciones más adelante. Durante la segunda guerra mundial se desarrolló una maquina cifradora llamada enigma, que fué utilizada para cifrar los mensajes secretos alemanes.

En general a lo largo de la historia la gente ha buscado maneras de esconder mensajes de tal manera que sólo las personas a los que van dirigidos puedan leerlos. Gran variedad de métodos se han desarrollado buscando este fin, unos más efectivos que otros, pero todos buscando el objetivo de la seguridad informática "Mantener la Privacidad, Integridad y Autenticidad".

Entre los algoritmos de encriptación más importantes se encuentran: el cifrado de Vernam, que se ha probado, tiene seguridad máxima pero su implementación es muy difícil; el DES, que fué durante años completamente seguro y el método de Diffie - Hellman, que intrudujo la llamada criptografía de llave pública, muy importante pues permite que dos personas se comunique de manera segura sin tener que intercambiar claves personalmente.

Por otro lado los autómatas celulares son mucho más recientes. Su historia puede dividirse en tres etapas principales (*wikipedia.org*): la era de Von Newman, la era de Martin Garder y la era de Stephen Wolfram.

Von Newman tenía la idea de crear una máquina capaz de autoreproducirse y, con la ayuda de Stanislav Ulam, implementó la teoría de los autómatas celulares en una rejilla cuadrada infinita que llamó espacio de evoluciones y a cada una de las posiciones, las llamó células. Cada célula cambiaba su estado, en tiempo discreto, dependiendo de los estados de sus células vecinas.

Siguiendo con la misma idea, en 1970, Jhon Conway creó el que probablemente ha sido el autómata celular más famoso de la historia; El Juego de la Vida. Este autómata, publicado en la revista *Scientific American* por Martin Gardner, fué todo un éxito. En los años siguientes se utilizó mucho tiempo de computador en todo el mundo para

buscar patrones que evolucionaran de manera particular.

Por último, Stephen Wolfram ha llevado a cabo numerosas investigaciones tratando de describir el comportamiento cualitativo de los autómatas celulares. Como resultados se muestran las siguientes clases:

- Clase I. La evolución lleva a una configuración estable y homogénea, es decir, todas las células terminan por llegar al mismo valor.
- Clase II. La evolución lleva a un cojunto de estructuras simples que son estables o periódicas.
- Clase III. La evolución lleva a un patrón caótico.
- Clase IV. La evolución lleva a estructuras aisladas que muestran un comportamiento complejo.

Entre los aportes de Wolfram se encuentra un artículo en el que se sugiere la utilización de un autómata con fines criptográficos. Dicho autómata tiene un comportamiento complejo que lo hace bastante bueno en la generación de sucesiones pseudo aleatorias para usos criptográficos.

Lo que se quiere con este proyecto es estudiar los autómatas celulares como generadores de sucesiones de *bits* pseudo aleatorias, realizando pruebas estadísticas que analizan sus propiedades criptográficas y determinan si un conjunto de sucesiones producidas por un mismo generador son, en su mayoría seguras para aplicaciones en criptografía.

Con los resultados obtenidos se desarrollará una aplicación criptográfica que utilice como base los autómatas celulares. Se trata de un ejemplo para mostrar que en realidad lo que se propone en el proyecto es viable. Sin embargo, no se pretende crear un sistema criptográfico como tal pues esto requiere de conocimientos muy avanzados, además de mucho tiempo de pruebas criptoanalíticas por parte de expertos en la materia que demuestren que el sistema es o no, realmente seguro.

Para el desarrollo del proyecto se escogió el método científico como la metodología investigativa más apropiada. Sin embargo, es necesario para los propositos del proyecto probar varias hipótesis. Aplicaremos una variación al método, la idea es proponer una hipótesis, aplicar las pruebas, evaluar los resultados y con base en estos proponer otra hipótesis y seguir el mismo procedimiento hasta encontrarnos con un resultado satisfactorio que sirva como base para la aplicación criptográfica.

1. PRESENTACIÓN DEL PROYECTO

1.1. PRESENTACIÓN DEL INFORME

El contenido de este documento se encuentra divido en capítulos, cada uno de ellos contiene información teórica necesaria para el desarrollo de la investigación.

El Capítulo 1 contiene la presentación del proyecto; objetivo general, objetivos específicos y justificación.

El Capítulo 2 presenta el marco teórico necesario para el desarrollo del proyecto. Incluye conceptos sobre Criptología, Autómatas Celulares, generadores de *bits* y pruebas estadísticas.

En el Capítulo 3 se muestra detalladamente la implementación de los generadores propuestos en el proyecto junto con pruebas estadísticas que validan los resultados.

El Capítulo 4 presenta la aplicación criptográfica de clave secreta utilizando los resultados obtenidos en el Capítulo 3.

El Capítulo 5 y 6 se encuentran las conclusiones y recomendaciones para desarrollos futuros.

A lo largo del libro se muestran figuras, sus fuentes son citadas en la parte inferior izquierda, sin embargo hay algunas que no contienen citas debido a que son hechas por los autores.

1.2. DESCRIPCIÓN DEL PROYECTO

1.2.1. Objetivo General

Estudiar los autómatas celulares unidimensionales de dos estados como generadores de cadenas de *bits* pseudo aleatorias, comprobar, mediante el criptoanálisis las propiedades criptográficas de éstas e implementar una aplicación criptográfica de clave secreta que las utilice.

1.2.2. Objetivos Específicos

- Estudiar los autómatas celulares unidimensionales de dos estados como generadores de cadenas de *bits* aleatorias, mediante la aplicación de pruebas estadísti-

cas.

- Comprobar, mediante el criptoanálisis las propiedades criptográficas de las cadenas de *bits* generadas por los autómatas celulares unidimensionales de dos estados.
- Implementar un generador de números pseudo aleatorios basado en autómatas celulares.
- Desarrollar una aplicación criptográfica de clave secreta, utilizando los autómatas celulares que presenten propiedades aleatorias y criptográficas.

1.3. JUSTIFICACIÓN

Actualmente, mantener secreta la información es una prioridad para la mayor parte de las personas y practicamente para todas las entidades, ya sean públicas o privadas. Imaginémonos por un momento un mundo en el que no hay comunicaciones confidenciales, cualquier persona podría saber cuales serán las estrategias de negocio de una empresa en los próximos anõs y si esa persona hace parte de la competencia seguramente tomaría medidas para evitar impactos indeseados. Debido a este tipo de situaciones la criptografía ha adquirido gran importancia en nuestro entorno ya que permite la transmisión de datos de manera confidencial entre dos personas que se comunican a través de una canal inseguro.

Un método criptográfico consiste en lo siguiente: de una clave secreta pequeña, generar una cadena pseudo aleatoria y con ésta cifrar la información. El problema es encontrar un algoritmo que aunque público, sea computacionalmente inviable descifrarlo. La cadena generada debe ser criptográficamente segura, para saber esto se puede llevar a cabo una comprobación de los postulados de Golomb ([8], pag 38), estos postulados, aunque no nos dicen con absoluta certeza si es criptográficamente segura, al menos nos brindan buenos indicios.

Aunque actualmente existen generadores de *bits* aleatorios que utilizan los autómatas celulares como base, se ha podido comprobar que no son muy seguros. W. Meier y O. Staffelbach en 1992 ([10]) pudieron, conociendo una parte del mensaje, obtener toda la clave secreta. Esto significa que podrían tener fácil acceso a todos los mensajes encriptados con ella.

El procedimiento es el siguiente : se conocen n valores de evolución de una célula (n valores de la sucesión cifrante) y se supone que ésta es la central. A continuación se generan de forma aleatoria los $n-1$ valores de las células que están a su derecha en el instante t. A partir de aquí, se determinan los valores de las células que aparecen en el

triángulo derecho de la configuración del Autómata Celular. Posteriormente, se deduce el valor de las células del triangulo izquierdo. Con ello se obtiene la configuración inicial completa que corresponde a la clave del criptosistema dado ([8]).

Se necesita cambiar la forma en la que se eligen las células que se usan como sucesión cifrante. En los algoritmos existentes, basados en los autómatas de Wolfram ([14]), se toman todos los valores consecutivos de la célula central como sucesión cifrante, esto es dar demasiada información, pensamos que al tomar un valor de cada dos generados, por ejemplo, el ataque propuesto por W. Meier y O. Staffelbach no sería exitoso.

Además usando otros autómatas celulares de dos estados diferentes a los 256 propuestos por Wolfram, podríamos tener mayor complejidad en los patrones obtenidos y una mayor dificultad al tratar de devolverse.

Con estas variaciones se espera encontrar una función que genere cadenas pseudo aleatorias que sean criptográficamente seguras, que puedan resistir los ataques contra los que son débiles los generadores existentes.

2. MARCO TEÓRICO

2.1. CRIPTOLOGÍA

La Criptología (del griego *criptos* = oculto y *logos* = tratado, ciencia) es el nombre con el que se designan dos disciplinas opuestas y a la vez complementarias: Criptografía y criptoanálisis. La criptografía se ocupa del diseño de algoritmos para cifrar, es decir, para enmascarar una determinada información de carácter confidencial. El criptoanálisis, por su parte, se ocupa de romper esos algoritmos de cifrado para así recuperar la información original. Ambas disciplinas se han desarrollado de forma paralela, pues cualquier método de cifrado lleva siempre emparejado su criptoanálisis correspondiente.

2.1.1. Criptografía

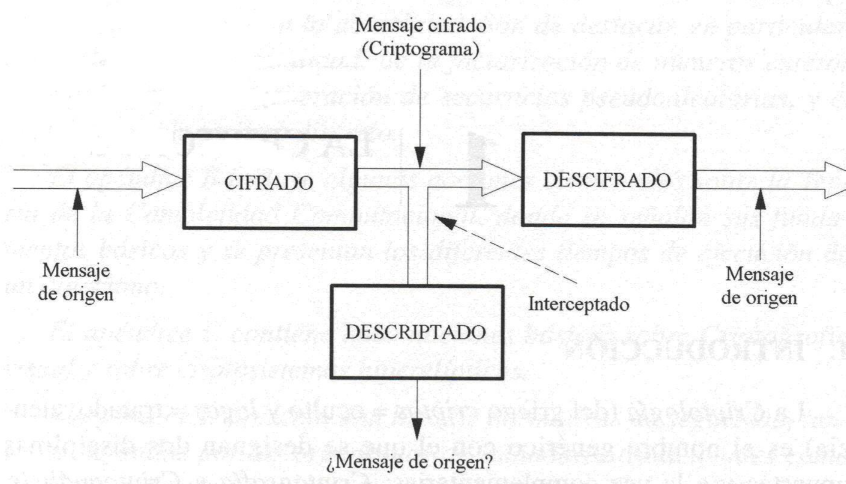
La criptografía como medio de proteger la información personal es un arte tan antiguo como la escritura. Como tal, permaneció durante siglos vinculada muy estrechamente a los círculos militares y diplomáticos, puesto que eran los únicos que en principio tenían necesidad de ella.

En la actualidad la situación ha cambiado drásticamente: el desarrollo de las comunicaciones electrónicas, unido al uso masivo y generalizado de los computadores, hace posible la transmisión y almacenamiento de grandes flujos de información confidencial que es necesario proteger. Es entonces cuando la criptografía pasa de ser una exigencia de minorías a convertirse en una necesidad real del hombre de la calle, que ve en esta falta de protección de sus datos privados una amenaza de su propia intimidad. El esquema fundamental de un proceso criptográfico (cifrado/descifrado) puede resumirse del modo en que se muestra en la figura 1.

A y B son, respectivamente, el emisor y receptor de un determinado mensaje. A transforma el mensaje original (texto claro o texto fuente), mediante un determinado procedimiento de cifrado controlado por una clave, en un mensaje cifrado (criptograma) que se envía por un canal público. En recepción, B con conocimiento de la clave transforma ese criptograma en el texto fuente, recuperando así la información original.

En el proceso de transmisión, el criptograma puede ser interceptado por un enemigo criptoanalista que lleva a cabo una labor de descriptado; es decir, intenta, a partir del criptograma y sin conocimiento de la clave, recuperar el mensaje original. Un buen sistema criptográfico será, por tanto, aquel que ofrezca un cifrado sencillo pero un des-

Figura 1: Proceso general cifrado/descifrado.



Tomado de [6]

criptado imposible o, en su defecto, muy difícil.

El tipo de transformación aplicada al texto claro o las características de las claves utilizadas marcan la diferencia entre los diversos métodos criptográficos. Una primera clasificación con base en las claves utilizadas puede desglosarse tal y como sigue:

- *Métodos simétricos*: Son aquellos en los que la clave de cifrado coincide con la de descifrado. Lógicamente, dicha clave tiene que permanecer secreta, lo que presupone que emisor y receptor se han puesto de acuerdo previamente en la determinación de la misma, o bien que existe un centro de distribución de claves que se la ha hecho llegar a ambos por un canal inseguro.

- *Métodos asimétricos*: Son aquellos en los que la clave de cifrado es diferente a la de descifrado. En general, la clave de cifrado es conocida libremente por el público, mientras que la de descifrado es conocida únicamente por el usuario.

Los métodos simétricos son propios de la criptografía clásica o criptografía de clave secreta, mientras que los métodos asimétricos corresponden a la criptografía de clave publica, introducida por Diffie y Hellman en 1976.[4]

Una de las diferencias fundamentales entre la criptografía clásica y moderna radica en el concepto de seguridad. Antes, los procedimientos de cifrado tenían una seguridad

probable; hoy, los procedimientos de cifrado han de tener una seguridad matemáticamente demostrable. Esto lleva a una primera clasificación de seguridad criptográfica: (ver [6])

- *Seguridad incondicional (teórica)*: el sistema es seguro frente a un atacante con un tiempo y recursos computacionales ilimitados. (ejemplo: cifrado de Vernam).

- *Seguridad computacional (práctica)*: el sistema es seguro frente a un atacante con tiempo y recursos limitados (ejemplo: sistemas de clave publica basados en problemas de alta complejidad de calculo).

- *Seguridad probable*: no se puede demostrar su integridad, pero el sistema no ha sido violado (ejemplo: Triple DES).

- *Seguridad condicional*: todos los demás sistemas, seguros en tanto que el enemigo carece de medios para atacarlos.

Con los antiguos procedimientos manuales y lentos de criptoanálisis era suficiente la seguridad condicional, pues en la mayoría de los casos se obtenía el descriptado del mensaje cuando la información del documento había perdido toda validez. Si el criptoanálisis tuvo éxitos de importancia fué solo por que, al igual que era lento el proceso de análisis, lo era también el cambio de claves. En la actualidad, con el uso de potentes computadores para el criptoanálisis, los operadores criptográficos tienen que tener propiedades matemáticas que los hagan invulnerables no solo en el presente, con nuestros conocimientos actuales de matemáticas y el estado actual de desarrollo de los computadores, sino también en un futuro a corto y mediano plazo.

- **Criptografía de clave secreta**

Los criptosistemas de clave secreta son aquellos en los que la clave es única tanto para el emisor como para el receptor, esto es, la clave con la que se cifra es idéntica a la clave con la que se descifra. Consecuentemente ésta debe mantenerse en secreto. Entre los criptosistemas simétricos podemos destacar dos tipos: los métodos de cifrado en flujo y los métodos de cifrado en bloque.

Veamos unos ejemplos de criptografia clásica

- Cifrado de Vigenère

  Es una generalización del cifrado del César, con la particularidad de que la clave toma sucesivamente diferentes valores. En términos matemáticos,

  $Y_i = X_i \oplus Z_i (mod\ 26)$,

  donde, por ejemplo, $Z_i = \{L, O, U, P\}$ alternativamente, siendo 26 el número de letras del alfabeto. Aunque este procedimiento de cifrado fué considerado seguro durante siglos, el Método Kasiski, publicado en 1863, consiguió romperlo (ver [6]).

- Cifrado de Vernam

  Representa el caso límite del cifrado de Vigenère en el que se emplea un alfabeto binario. La operación aritmética es una suma módulo 2 (equivalente a una función *xor*), y la clave una sucesión aleatoria binaria de la misma longitud del texto claro. Para recuperar el mensaje original se suma nuevamente al criptograma la sucesión aleatoria, ya que la suma módulo 2 es una involución. Este método de cifrado fué utilizado durante la Segunda Guerra Mundial y se creyó durante mucho tiempo que su seguridad criptográfica era total, pero Shannon no dió una prueba de la misma hasta 1946 (para más detalles véase [6]). Aunque este método ofrece una seguridad máxima, resulta en muchas ocasiones inviable computacionalmente, ya que se requiere un dígito de clave secreta por cada dígito de texto claro, con lo que si la información a cifrar es muy elevada necesitamos generar una sucesión aleatoria de muchos dígitos, que además debe ser la clave que conozcan el emisor y el receptor. Los métodos de cifrado en flujo, que se desarrollan en apartados posteriores, se basan en este método.

- **Criptosistemas de Cifrado en Flujo**

Consisten en transformar el texto claro (escrito como una cadena de *bits*) mediante la suma *bit* a *bit* de una sucesión pseudo aleatoria de *bits*, dando lugar así al criptograma. Por lo tanto, la idea fundamental del cifrado en flujo consiste en generar una sucesión larga e imprevisible de dígitos binarios a partir de una clave corta elegida de forma aleatoria. Este método es tanto más seguro cuanto más se aproxime la sucesión binaria generada a una auténtica sucesión aleatoria.

Recordemos, que aunque el método Vernam ofrecía las máximas garantías de seguridad, presentaba el incoveniente de necesitar una gran capacidad computacional, sobre todo a medida que la cantidad de información a cifrar aumenta ya que se necesita un *bit* de de clave secreta por cada *bit* de texto claro. Por ello, quedó reservado a circunstancias en las que se requiere condiciones de máxima seguridad en textos, pero un mínimo de información a proteger.

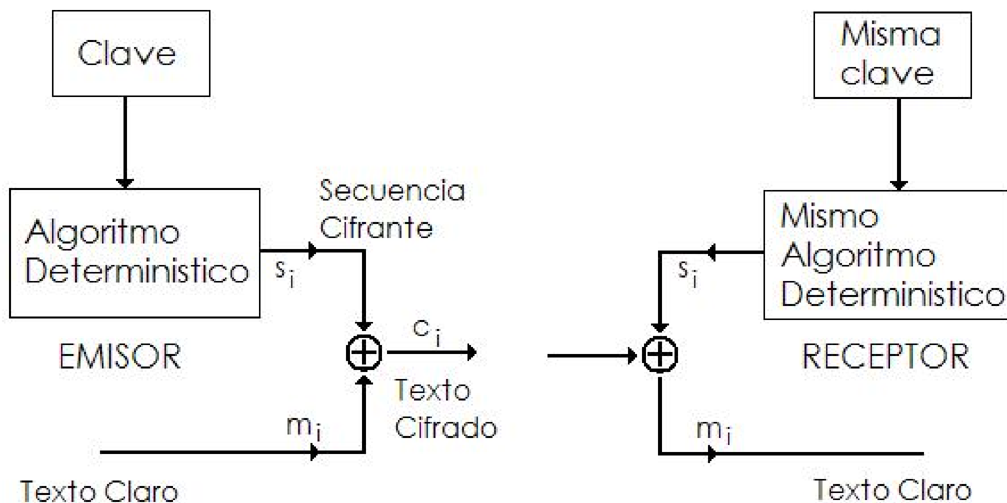
En la práctica lo que se hace es utilizar algoritmos determinísticos que, a partir de una clave corta conocida sólo por el emisor y el receptor, generen simultáneamente en emisión y en recepción una sucesión de la longitud deseada. Esta será la sucesión cifrante que se suma módulo 2 con el texto claro en el cifrado, o con el criptograma en el descifrado, y que hace las veces de clave en el cifrado Vernam.

Es decir, el método de cifrado en flujo no es más que una aproximación al cifrado Vernam, tanto más segura cuanto más se aproxime la sucesión binaria generada a una auténtica sucesión aleatoria. Utilizaremos sucesiónes para cifrar de la misma longitud que el texto claro.

Este método es mucho más económico porque la clave a utilizar es lo único secreto que deben conocer el emisor y el receptor (el algoritmo se asume público), ya que al tratarse de algoritmos determinísticos, la sucesión que genere dicha clave será la misma en la recepción y en la emisión. Dicha clave es mucho más pequeña que la sucesión cifrante, y por lo tanto la cantidad de información a intercambiar disminuye.

Así el método consistiría en lo siguiente: el emisor con una clave secreta de poca longitud y un algoritmo determinístico (público), genera una sucesión binaria de longitud n, que es la denominada sucesión cifrante, $S = \{s_i\}$, con $1 \leq i \leq n$ y $s_i \in \{0, 1\}$, cuyos elementos se suman módulo 2 con los correspondientes *bits* de *texto claro* $M = \{m_i\}$, con $1 \leq i \leq n$, dando logar a los *bits* de *texto cifrado* o *criptograma* $C = \{c_i\}$, con $1 \leq i \leq n$ que es la que se envía a través de un canal público.

Figura 2: Esquema General de Cifrado en Flujo



Tomado de [6]

En recepción, con la misma clave y el mismo algoritmo determinístico, genera la misma sucesión cifrante S, que se suma módulo 2 con la sucesión cifrada C, dando lugar a los *bits* de texto claro M. El procedimiento de cifrado y descifrado es idéntico, ya que, como sabemos, la suma módulo 2 es una involución.

Como la sucesión cifrante se ha obtenido a partir de un algoritmo determinístico, el cifrado en flujo ya no considera sucesiónes perfectamente aleatorias, sino solamente pseudo aleatorias. Sin embargo, lo que se pierde en cuanto a seguridad, se gana en viabilidad práctica a la hora de utilizar este procedimiento de cifrado.

La sucesión generada para realizar el cifrado debe cumplir una serie de requisitos para poder garantizar una cierta seguridad:

1. El periodo de la sucesión cifrante ha de ser al menos tan largo como la longitud del texto a cifrar.
2. Diferentes muestras de una determinada longitud han de estar uniformemente distribuidas a lo largo de toda la sucesión, con lo que aseguramos que las propiedades de aleatoriedad de la sucesión son buenas. Para ello, como veremos más adelante, nos basamos en los Postulados de Golomb.
3. La sucesión cifrante ha de ser imprevisible; es decir, dada una porción de sucesión de cualquier longitud, no se puede predecir el siguiente dígito con una probabilidad de acierto superior a 1/2.
4. La sucesión tiene que ser fácil de generar con medios electrónicos para su aplicabilidad en el proceso real de cifrado/descifrado. Además, hemos de tener en cuenta una serie de aspectos técnicos: velocidad de generación de la sucesión, coste, tamaño, etc.

- Ejemplo de cifrado en flujo: **Algoritmo RC4**

El Algoritmo RC4 fué diseñado por Ron Rivest en 1987 para la compañía RSA Data Security. Su implementación es extremadamente sencilla y rápida, y está orientado a generar sucesiónes en unidades de un *byte*, además de permitir claves de diferentes longitudes. Por desgracia es un algoritmo propietario, lo cual implica que no puede ser incluido en aplicaciones de tipo comercial sin pagar los derechos correspondientes. El código del algoritmo no se ha publicado nunca oficialmente, pero en 1994 alguien difundió en los grupos de noticias de Internet una descripción que, como posteriormente se ha comprobado, genera las mismas sucesiónes[9]. Dicha descripción consta de una S-Caja de 8*8, que almacenará una permutación del conjunto $\{0, ..., 255\}$. Dos contadores i y j se ponen en cero. Luego, cada *byte* O_r de la sucesión se calcula como sigue:

1. $i = (i+1) \bmod 256$
2. $j = (j + S_i) \bmod 256$
3. Intercambiar los valores de S_i y S_j
4. $t = (S_i + S_j) \bmod 256$
5. $O_r = S_t$

Para calcular los valores iniciales de la S-Caja, se hace lo siguiente:

1. $S_i = i \; \forall \; 0 \leq i \leq 255$
2. Rellenar el vector K_0 a $K_2 55$ repitiendo la clave tantas veces como sea necesario
3. $j = 0$
4. Para $i = 0$ hasta 255 hacer:

    $j = (j + S_i + K_i) \bmod 256$

    Intercambiar S_i y S_j

El algoritmo RC4 genera sucesiónes en las que los ciclos son bastante grandes, y es inmune a los criptoanálisis diferencial y lineal, si bien algunos estudios indican que puede poseer claves débiles, y que es sensible a estudios analíticos del contenido de la S-Caja. De hecho, algunos afirman que en una de cada 256 claves posibles, los *bytes* que se generan tienen una fuerte correlación con un subconjunto de los *bytes* de la clave, lo cual es un comportamiento muy poco recomendable.

A pesar de las dudas que existen en la actualidad sobre su seguridad, es un algoritmo ampliamente utilizado en muchas aplicaciones de tipo comercial.

- **Criptosistemas de Cifrado en Bloque**

En éstos, el mensaje claro se rompe en una serie de bloques de igual tamaño, siendo cifrado cada uno de ellos haciendo uso del criptosistema, por ejemplo, DES, IDEA, Triple-DES, Rijndael, etc.

Como características de un método de cifrado en bloque podemos señalar las siguientes: cada símbolo se cifra dependiendo de los símbolos adyacentes; cada bloque de símbolos, independientemente de su posición, se cifra de la misma manera; si la clave es la misma, al cifrar un mensaje original se obtiene siempre el mismo mensaje cifrado; y podemos descifrar los bloques que nos interesen sin necesidad de descifrar todo el mensaje.

Así, todos los cifrados en bloques se basan en cuatro elementos:

1. El primero de ellos es la transformación inicial, que puede consistir simplemente en “descolocar” los símbolos de la sucesión inicial, o que puede tener significación criptográfica, en cuyo caso dependería de una clave.

2. El segundo de los elementos es una función criptográfica, generalmente de carácter débil, iterada un número determinado de veces. Normalmente consiste en una función no lineal dependiente de los datos y de la clave, y puede estar formada por una única operación muy compleja o por la sucesión de varias transformaciones simples. Las sucesivas iteraciones se enlazan mediante sumas módulo 2 *bit* a *bit* con los datos de la sucesión inicial o de iteraciones anteriores, lo que permite involuciones a la hora del criptoanálisis.

3. El tercer elemento es la transformación final que sirve para que las operaciones de encriptación y desencriptación sean simétricas.

4. El cuarto elemento es un algoritmo de expansión de clave que sirve para convertir la clave del usuario en un conjunto de subclaves.

- Ejemplo de cifrado en bloque: **Algoritmo DES**

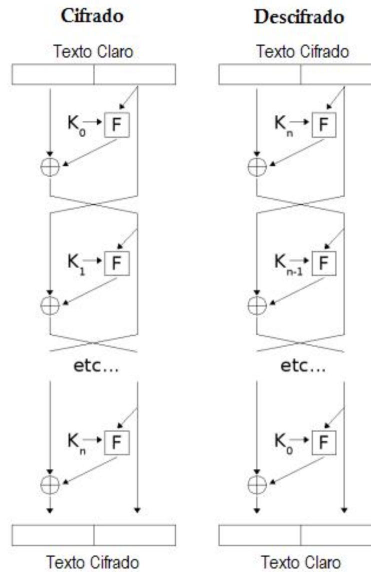
Es el algoritmo simétrico más extendido mundialmente. Se basa en el algoritmo LUCIFER, que había sido desarrollado por IBM a principios de los setenta, y fué adoptado como estándar por el Gobierno de los EE.UU. para comunicaciones no clasificadas en 1976. En realidad la NSA lo diseñó para ser implementado por hardware, creyendo que los detalles iban a ser mantenidos en secreto, pero la Oficina Nacional de Estandarización publicó su especificación con suficiente detalle como para que cualquiera pudiera implementarlo por software. No fué casualidad que el siguiente algoritmo adoptado (*Skipjack*) fuera mantenido en secreto.

A mediados de 1998, se demostró que un ataque por la fuerza bruta a DES era viable, debido a la escasa longitud que emplea en su clave. No obstante, el algoritmo aún no ha demostrado ninguna debilidad grave desde el punto de vista teórico, por lo que su estudio sigue siendo plenamente interesante.

El algoritmo DES codifica bloques de 64 *bits* empleando claves de 56 *bits*. Es una Red de Feistel (ver Figura 3) de 16 rondas, más dos permutaciones, una que se aplica al principio (P_i) y otra que se aplica al final (P_f), tales que $P_i = P_f^{-1}$.

La función f (figura 4) se compone de una permutación de expansión (E), que convierte el bloque de 32 *bits* correspondiente en uno de 48. Después realiza un $or-exclusivo$ con el valor K_i, también de 48 *bits*, aplica ocho $S-Cajas$ de 6*4 *bits*, y

Figura 3: Red de Feistel



Tomado de Wikipedia.org

efectua una nueva permutación P.

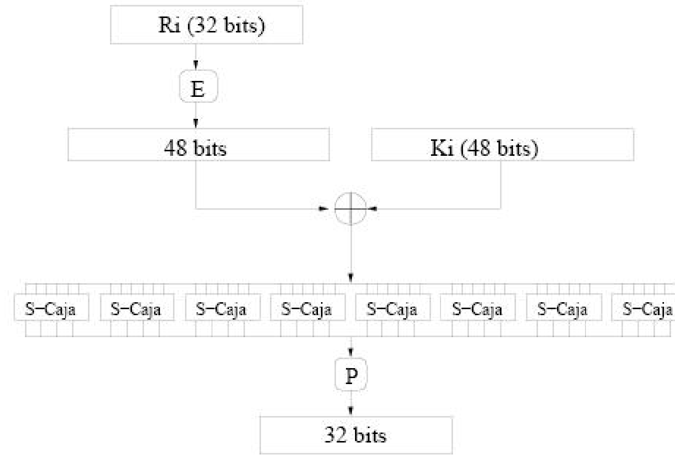
Se calculan un total de 16 valores de K_i (figura 5), uno para cada ronda, efectuando primero una permutación inicial **EP1** sobre la clave de 64 *bits*, llevando a cabo desplazamientos a la izquierda de cada una de las dos mitades –de 28 *bits*– resultantes, y realizando finalmente una elección permutada (**EP2**) de 48 *bits* en cada ronda, que será la K_i. Los desplazamientos a la izquierda son de dos *bits*, salvo para las rondas 1, 2, 9 y 16, en las que se desplaza sólo un *bit*. Nótese que aunque la clave para el algoritmo DES tiene en principio 64 *bits*, se ignoran ocho de ellos –un *bit* de paridad por cada *byte* de la clave–, por lo que en la práctica se usan sólo 56 *bits*.

Para descifrar basta con usar el mismo algoritmo (ya que $P_i = P_f^{-1}$) empleando las K_i en orden inverso.

- **Criptografía de clave pública**

Los algoritmos de llave pública, o algoritmos asimétricos, han demostrado su interés para ser empleados en redes de comunicación inseguras (Internet). Introducidos por Whitfield Diffie y Martin Hellman a mediados de los años 70, su novedad fundamental con respecto a la criptografía simétrica es que las claves no son únicas, sino que forman

Figura 4: Esquema de la funcion f del algoritmo DES



Tomado de [9]

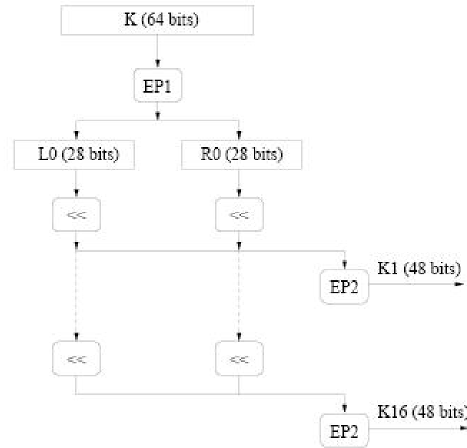
pares. Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros; otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original o porque la longitud de la clave es enorme. Se basan en general en plantear al atacante problemas matemáticos difíciles de resolver. En la práctica muy pocos algoritmos son realmente útiles. El más popular por su sencillez es RSA, que ha sobrevivido a multitud de ataques, si bien necesita una longitud de clave considerable. Otros algoritmos son los de ElGamal y Rabin.

Los algoritmos asimétricos emplean generalmente longitudes de clave mucho mayores que los simétricos. Por ejemplo, mientras que para algoritmos simétricos se considera segura una clave de 128 *bits*, para algoritmos asimétricos –si exceptuamos aquellos basados en curvas elípticas– se recomiendan claves de al menos 1024 *bits*. Además, la complejidad de cálculo que comportan estos últimos los hace considerablemente más lentos que los algoritmos de cifrado simétricos. En la práctica los métodos asimétricos se emplean únicamente para codificar la clave de sesión (simétrica) de cada mensaje o transacción particular.

- **Aplicaciones de los Algoritmos Asimétricos**

Los algoritmos asimétricos poseen dos claves diferentes en lugar de una, K_p y K_P, denominadas *clave privada* y *clave pública*. Una de ellas se emplea para codificar, mientras que la otra se usa para decodificar. Dependiendo de la aplicación que le demos al algoritmo, la clave pública será la de cifrado o viceversa. Para que estos criptosistemas sean seguros también ha de cumplirse que a partir de una de las claves resulte extremadamente difícil calcular la otra.

Figura 5: Cálculo de las K para el algoritmo DES.



Tomado de [9]

- Protección de la Información

  Una de las aplicaciones inmediatas de los algoritmos asimétricos es el cifrado de la información sin tener que transmitir la clave de decodificación, lo cual permite su uso en canales inseguros. Supongamos que **A** quiere enviar un mensaje a **B** (figura 6). Para ello solicita a **B** su clave pública K_P . **A** genera entonces el mensaje cifrado $E_{K_P}(m)$. Una vez hecho esto únicamente quien posea la clave K_p –en nuestro ejemplo, B– podrá recuperar el mensaje original m.

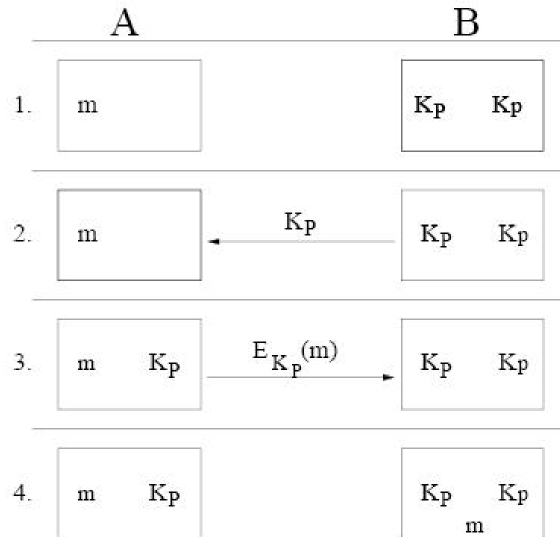
  Nótese que para este tipo de aplicación, la llave que se hace pública es aquella que permite codificar los mensajes, mientras que la llave privada es aquella que permite descifrarlos.

- Autenticación

  La segunda aplicación de los algoritmos asimétricos es la autenticación de mensajes, con ayuda de funciones resumen que nos permiten obtener una firma digital a partir de un mensaje. Dicha firma es mucho más pequeña que el mensaje original, y es muy difícil encontrar otro mensaje que de lugar a la misma. Supongamos que **A** recibe un mensaje m de **B** y quiere comprobar su autenticidad. Para ello **B** genera un resumen del mensaje $r(m)$ (ver figura 7) y lo codifica empleando la clave de cifrado, que en este caso será privada.

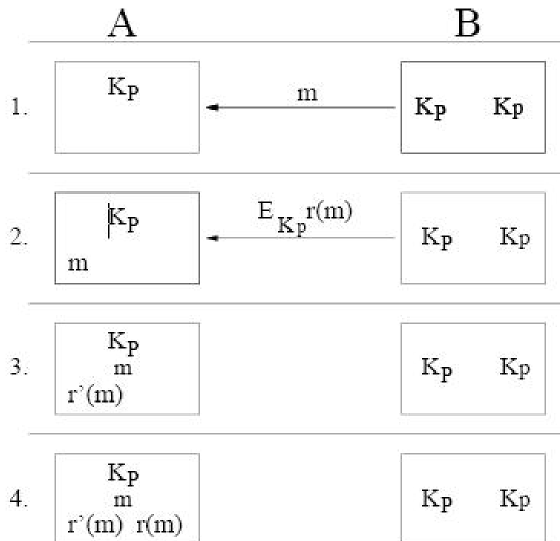
  La clave de descifrado se habrá hecho pública previamente, y debe estar en poder de **A**. **B** envía entonces a **A** el criptograma correspondiente a $r(m)$. A puede ahora generar su propia $r'(m)$ y compararla con el valor $r(m)$ obtenido del criptograma enviado por **B**. Si coinciden, el mensaje será auténtico, puesto que el único que

Figura 6: Transmisión de la información usando algoritmos asimétricos



Tomado de [9]

Figura 7: Autenticación de información empleando algoritmos asimétricos.



Tomado de [9]

posee la clave para codificar es precisamente B.

Nótese que en este caso la clave que se emplea para cifrar es la clave privada, justo al revés que para la simple codificación de mensajes.

En muchos de los algoritmos asimétricos ambas claves sirven tanto para cifrar como para descifrar, de manera que si empleamos una para codificar, la otra permitirá decodificar y viceversa. Esto ocurre con el algoritmo RSA, en el que un único par de claves es suficiente para codificar y autenticar.

- **Algoritmo Diffie - Hellman**

Es un algoritmo asimétrico, basado en el problema de Diffie-Hellman, que se emplea fundamentalmente para acordar una clave común entre dos interlocutores, a través de un canal de comunicación inseguro. La ventaja de este sistema es que no son necesarias llaves públicas en el sentido estricto, sino una información compartida por los dos comunicantes.

Sean **A** y **B** los interlocutores en cuestión. En primer lugar, se calcula un número primo p y un generador α de Z_p^* con $2 \leq \alpha \leq p-2$. Esta información es pública y conocida por ambos. El algoritmo queda como sigue:

1. **A** escoge un número pseudo aleatorio x, comprendido entre 1 y $p-2$ y envía a **B** el valor α^x $(mod\ p)$.
2. **B** escoge un número pseudo aleatorio y, análogamente al paso anterior, y envía a **A** el valor α^y $(mod\ p)$.
3. **B** recoge α^x y calcula $K = (\alpha^x)^y\ mod(p)$.
4. **A** recoge α^y y calcula $K = (\alpha^y)^x\ mod(p)$.

Puesto que x y y no viajan por la red, al final **A** y **B** acaban compartiendo el valor de K, sin que nadie que capture los mensajes transmitidos pueda repetir el cálculo.

- **Funciones Hash**

Una *Función Hash* es una función matematica o de otra clase, que toma una entrada de longitud variable (llamada pre-imagen) y la convierte en una cadena de salida de longitud fija (generalmente más pequeña llamada *código hash*).

La idea de las funciones hash es tomar una “huella digital” de la pre-imagen; de tal manera que podamos saber si el mensaje después de ser desencriptado es el mismo que cuando fué encriptado con una probabilidad bastante alta. Es importante resaltar que las funciones hash son del tipo muchos a uno, entonces, no podemos determinar con absoluta certeza que dos pre-imágenes son iguales sólo porque su código hash es el mismo.

Una *función hash de un solo sentido* es aquella en la que es fácil calcular el código hash teniendo la pre-imagen, pero es muy difícil hacer una pre-imagen que tenga un código hash específico. Ademas una buena función hash de un solo sentido es también resistente a las colisiones, de tal manera que es muy difícil crear dos pre-imágenes que tengan el mismo código hash.

La función hash es pública, no hay nada secreto en el proceso. La seguridad de ésta se encuentra en su caracteística de un solo sentido. La salida no es dependiente de la entrada en ninguna manera fácilmente identificable. Un cambio de un solo *bit* en la pre-imagen cambiará en promedio, la mitad de los *bits* en el código hash. Esto es particularmente útil para poder preservar la integridad de los mensajes enviados. Digamos por ejemplo que necesito enviar un mensaje encriptado, si sólo mando el texto cifrado, entonces, cualquier persona podría interceptar el mensaje y cambiarlo, después de esto el receptor del mensaje lo decodifica y lee un mensaje diferente al que fué enviado. Para evitar esta situación lo que se hace es calcular el código hash del mensaje antes de encriptarlo y enviarlo, en texto claro, junto con el mensaje encripado. Así, cuando el receptor tenga el mensaje, puede descifrarlo y verificar que es el mensaje original comparando los dos codigos hash (el que iba con el texto crifrado y el que calculó él del mensaje desencriptado).

- **Función Hash MD5**

Esta función es una de las más usadas para criptografia, sin embargo su popularidad ha venido en decadencia desde que se logró su criptoanálisis haciéndola vulnerable a ciertos tipos de ataques.

Descripción del algoritmo:

Después de un procesamiento inicial, MD5 procesa la pre-imagen en bloques de 512 *bits* divididos en 16 sub-bloques de 32 *bits* cada uno. La salida del algoritmo (código hash) cosiste en un conjunto de cuatro bloques de 32 *bits* que se concatenan para formar el código hash de 128 *bits*.

Primero la pre-imagen se alarga de tal manera que su longitud se exactamente 64 *bits* inferior a un múltiplo de 512. Éste proceso consiste en concatenar un uno al final

de la entrada seguido de tantos ceros como sean necesarios. Después, la longitud de la pre-imagen inicial (antes de añadirle el uno y los ceros) es concatenada como un entero de 64 *bits* al final del archivo. Esto se hace para que la longitud del mensaje sea exactamente múltiplo de 512 (requerido para el resto del proceso), mientras que se asegura que diferentes mensajes no quedarán iguales después del alargamiento.

Se inicializan cuatro variables de 32 *bits*:

$$\begin{aligned} A &= 0x01234567 \\ B &= 0x89abcdef \\ C &= 0xfedcba98 \\ D &= 0x76543210 \end{aligned}$$

Posteriormente comienza el lazo principal del algoritmo, que se repetirá para cada bloque de 512 *bits* del mensaje. En primer lugar copiaremos los valores de A,B,C y D en otras cuatro variables, a,b,c y d. Luego definiremos las siguientes cuatro funciones:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\ G(X,Y,Z) &= (X \wedge Y) \vee (Y \wedge (\neg Z)) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee (\neg Z)) \end{aligned}$$

Estas funciones están diseñadas para que cada unos de los *bits* correspondentes a X, Y y Z sean independientes y no sesgados, entonces, cada *bit* del resultado tendrá las mismas características.

Ahora, si M_j representa el j-ésimo sub-bloque del mensaje, y $<<< s$ significa desplazar circularmente la representación binaria del valor s *bits* la izquierda, las cuatro operaciones son:

$$\begin{array}{lll} FF(a,b,c,d,M_j,s,t_i) & \text{denota} & a = b + ((a + F(b,c,d) + M_j + t_i) <<< s) \\ GG(a,b,c,d,M_j,s,t_i) & \text{denota} & a = b + ((a + G(b,c,d) + M_j + t_i) <<< s) \\ HH(a,b,c,d,M_j,s,t_i) & \text{denota} & a = b + ((a + H(b,c,d) + M_j + t_i) <<< s) \\ II(a,b,c,d,M_j,s,t_i) & \text{denota} & a = b + ((a + I(b,c,d) + M_j + t_i) <<< s) \end{array}$$

Las funciones FF, GG, HH e II son aplicadas en cada una de las cuatro rondas del algoritmo. después de hacer todo esto los valores correspondientes a *a, b, c* y *d* se suman con *A, B, C, D* respectivamente. El producto final es la concatenación de *A, B, C,* y *D.*

2.1.2. Criptoanálisis

El *Criptoanálisis* consiste en romper la seguridad de un criptosistema. Esto se puede lograr de dos maneras, la primera es descifrar un mensaje sin conocer la llave y la segunda es encontrar la clave a partir de uno o más criptogramas que han sido encriptados con ésta.

Criptoanálisis de Algoritmos Simétricos

Se podría decir que el criptoanálisis se comenzó a estudiar seriamente con la aparición de DES. Mucha gente desconfiaba (y aún desconfía) del algoritmo propuesto por la NSA (*National Security Agency*). Se dice que existen estructuras extrañas, que muchos consideran sencillamente puertas traseras colocadas por la Agencia para facilitarles la decodificación de los mensajes. Nadie ha podido aún demostrar ni desmentir este punto. Lo único cierto es que el interés por buscar posibles debilidades en él ha llevado a desarrollar técnicas que posteriormente han tenido éxito con otros algoritmos.

Estos mecanismos son significativamente más eficientes que la fuerza bruta para criptoanalizar un mensaje. Los dos métodos que vamos a comentar parten de que disponemos de grandes cantidades de pares texto claro-texto cifrado obtenidos con la clave que queremos descubrir.

- **Criptoanálisis Diferencial**

  Descubierto por Biham y Shamir en 1990, permite efectuar un ataque de texto claro escogido a DES que resulta más eficiente que la fuerza bruta. Se basa en el estudio de los pares de criptogramas que surgen cuando se codifican dos textos claros con diferencias particulares, analizando la evolución de dichas diferencias a lo largo de las rondas de DES. Para llevar a cabo un criptoanálisis diferencial se toman dos mensajes cualesquiera (incluso aleatorios) idénticos salvo en un número concreto de *bits*. Usando las diferencias entre los textos cifrados, se asignan probabilidades a las diferentes claves de cifrado. Conforme tenemos más y más pares, una de las claves aparece como la más probable. Esa será la clave buscada.[9]

- **Criptoanálisis Lineal**

  El criptoanálisis lineal, descubierto por Mitsuru Matsui, basa su funcionamiento en tomar algunos *bits* del texto claro y efectuar una operación *xor* entre ellos, tomar algunos del texto cifrado y hacerles lo mismo, y finalmente hacer un *xor* de los dos resultados anteriores, obteniendo un único *bit*. Efectuando esa operación a una gran cantidad de pares de texto claro y criptograma diferentes podemos ver si

se obtienen más ceros o más unos. Si el algoritmo criptográfico en cuestión es vulnerable a este tipo de ataque, existirán combinaciones de *bits* que, bien escogidas, den lugar a un sesgo signićativo en la medida anteriormente definida, es decir, que el número de ceros (o unos) es apreciablemente superior.[9] Esta propiedad nos va a permitir poder asignar mayor probabilidad a unas claves sobre otras y de esta forma descubrir la clave que buscamos.

Otros tipos de ataques a los que son vulnerables los cifradores en flujo se muestran a continuacion:

- **Clave Reutilizada**

  Los Cifradores en flujo son vulnerables a ataques si la misma clave es usada más de una vez. Digamos por ejemplo que se envían los mensajes A y B de la misma longitud, y ambos encriptados usando la misma clave, K. La sucesión cifrante generada será entonces $C(K)$ y tendrá la misma longitud de los mensajes. Las versiones encriptadas de los mensajes serán:

  $$E(A) = A \; xor \; C$$
  $$E(B) = B \; xor \; C$$

  Donde la función *xor* es aplicada *bit* por *bit*.

  Supongamos ahora que un atacante intercepta los dos mensajes ecriptados. Él puede calcular fácilmente:

  $$E(A) \; xor \; E(B)$$

  Además como la función *xor* es conmutativa y tiene la propiedad de $X \; xor \; X = 0$ (autoinversa) entonces:

  $$E(A) \; xor \; E(B) = (A \; xor \; C) \; xor \; (B \; xor \; C) = A \; xor \; B \; xor \; C \; xor \; C = A \; xor \; B$$

  Si A y B están escritos en leguaje natural, la expresion $A \; xor \; B$ es un tipo de cifrador (conocido en inglés como *Running Key Cipher*) que es muy inseguro. Este cifrador puede romperse con métodos manuales (a láliz y papel) en sólo cuestión de minutos.

  En general, si un mensaje es más largo que el otro entonces el atacante deberá cortarlo de tal manera que queden de la misma longitud, en este caso el mensaje más

largo no podrá ser recuperado en su totalidad.

Una forma de evitar este problema es utilizar un vector de inicialización (V.I.) que se combina con la clave principal para crear una clave de un solo uso. Este mecanismo es usado en varios criptosistemas que utilizan el popular cifrador en flujo *RC4*, incluyendo *Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA)* y *Ciphersaber*. Uno de los problemas con el algoritmo *WEP* era que tenía su V.I. muy corto, solo 24 *bits*, esto significaba que era muy probable que la clave se repitiera si se mandaban algunos miles de paquetes, haciendo que aquellos con el V.I. duplicado fueran vulnerables al ataque de clave reutilizada.

- **Ataque por Sustitución**

  Supongamos que un atacante conoce exactamente parte o todo el contenido de uno de nuestros mensajes. Como parte del ataque del *Man in the Middle* (Hombre en la Mitad), él puede alterar el contenido del mensaje sin saber la clave K. Digamos por ejemplo que nuestro mensaje contiene la cadena “$5000,00”, el atacante podría cambiar eso por “$3456,90” *xoreando* esa parte del texto cifrado con la cadena “$5000,00” *xor* “$3456,90”. Para ver como funciona supongamos que nuestro mensaje cifrado es solo $C(K)$ *xor* “$5000,00”. lo que el atacante estaría haciendo sería:

$$C(K) \text{ } xor \text{ “\$5000,00” } xor \text{ “\$5000,00” } xor \text{ “\$3456,90” } = C(K) \text{ } xor \text{ “\$3456,90”}$$

  Que sería lo que nuestro texto cifrado contendría si la cantidad correcta fuera “$3456,90”. Este tipo de ataque puede ser prevenido utilizando un sistema de autenticación de mensajes para aumentar la probabilidad de que los cambios hechos en el texto cifrado sean detectados.

2.2. generadorES DE BITS PSEUDO ALEATORIOS

Aunque existen generadores de números aleatorios (como los presentados en [9] pag. 93), los generadores de números y *bits* pseudo aleatorios son más utilizados debido a su fácil implementación y a que son repetibles, esto quiere decir que dada una misma entrada, la salida es igual. Ahora enfocándonos en la generación de cadenas de *bits*, se puede decir lo siguiente:
Un *generador de bits pseudo aleatorios* (*PRNG* por sus siglas en inglés *Pseudo Random Number Generator*) es un algoritmo determinístico al que, proporcionándole una sucesión binaria realmente aleatoria de longitud k, produce una sucesión binaria de longitud

$l >> k$ que "parece" ser aleatoria.([8], Pag. 29)

Por lo dicho anteriormente es claro que la salida de estos generadores no es aleatoria, sin embargo eso no es necesario. Lo que se quiere es producir sucesiones que parezcan aleatorias de tal manera que sea muy dificil distinguir entre la salida del generador y una sucesion de *bits* realmente aleatoria, es decir, que el tiempo requerido para este procedimiento sea tan grande que lo haga inviable.

Se llama *semilla* a la entrada de longitud k de un generador pseudo aleatorio y *sucesión pseudo aleatoria* (ya que no es aleatoria) a la salida de longitud l.(ver Fig. 8) Cabe anotar que el número de posibles sucesiones pseudo aleatorias de salida es una pequeña fracción del orden de $2^k/2^l$ de todas las posibles sucesiones binarias de longitud l.([8], Pag. 29)

Figura 8: Generación de *bits* Pseudo Aleatorios

Para tener alguna garantía de que las sucesiones generadas no son previsibles y por lo tanto son fiables, se implementan una serie de pruebas estadísticas diseñadas para estudiar ciertas propiedades que un generador aleatorio debe tener. Además, si pretendemos usar las sucesiones pseudo aleatorias producidas para fines criptográficos es necesario que satisfagan unos mínimos requisitos de seguridad entre los que podemos citar como fundamental el siguiente: la longitud k de la sucesión aleatoria de entrada debe ser lo suficientemente grande como para que una búsqueda exhaustiva (criptoanálisis por fuerza bruta) realizada sobre las 2^k posibles sucesiones requiera un tiempo de ejecución no factible.

Dos son pues, los requerimientos que deben tener las sucesiones producidas por generadores pseudo aleatorios para que sean estadísticamente indistinguibles (en un tiempo de cómputo razonable) de verdaderas sucesiones aleatorias: deben tener propiedades aleatorias (lo cual queda demostrado si pasan las diferentes pruebas estadísticas) y la sucesión de entrada debe tener una longitud suficiente como para asegurarnos que una búsqueda exaustiva no sea viable. Esto se pone de manifiesto en:

Un generador de *bits* pseudo aleatorios se dice que pasa todas las pruebas estadísticas en tiempo polinómico si no existe un algoritmo de tiempo polinómico (esto es, el tiempo de ejecución del mismo está acotado por un polinomio en la longitud de la sucesión de salida) que pueda distinguir entre la salida del generador y una sucesión

“realmente” aleatoria de la misma longitud, con probabilidad significativamente mayor que $\frac{1}{2}$.

2.2.1. Ejemplos de generadores de *bits*

- **El generador lineal congruencial**

  Este es uno de los más difundidos siendo a la vez uno de los más débiles y no apto para uso criptográfico. Se utiliza en la mayoría de las funciones *rand()* y similares. La forma general de es: $X_{i+1} = (A \cdot X_i + b)\, mod\ m$ donde X_0 es la semilla del generador.[9]

  Propiedades

  1. El módulo m es una cota superior del número de valores diferentes que puede tomar la semilla.
  2. Si X_i vuelve a tomar el valor de la semilla inicial la sucesión se repetirá cíclicamente.
  3. Todas las sucesiónes producidas por este tipo de generador si se prolongan lo suficiente (periodo) acaban en un ciclo que se repite indefinidamente.

- **El generador ANSI X9.17**

  Este generador esta pensado como mecanismo para generar claves DES y vectores de inicialización, usando *triple - DES* como primitiva (podría usarse otro algoritmo de cifrado de bloques). También es ampliamente usado en banca y otras aplicaciones.

  1. K es una clave secreta para *triple - DES* generada de algún modo en el momento de la inicialización. Debe ser aleatoria y usada sólo para este generador. Es parte del estado secreto del *PRNG* y nunca varía con las entradas.[9]
  2. Cada vez que se desee una salida se hace lo siguiente :
     a) $T_i = E_K(timestamp)$
     b) $salida[i] = E_K(T_{i-1}\ xor\ semilla[i])$
     c) $semilla[i+1] = E_K(T_{i-1}\ xor\ salida[i])$

     El *timestamp* se basa en el uso de la CPU del programa y su resolución depende de la plataforma utilizada, por ejemplo en Linux tiene una resolución de 0.01 segundos.

- **El generador de DSA**

DSA (*Digital Signature Standard*) describe un PRNG basado en SHA[5] pensado para generar parámetros pseudo aleatorios para el algoritmo de firmas DSA. Este generador permite entradas opcionales del usuario, W_i, mientras se generan las claves. De otro modo el PRNG nunca se recobraría de un estado interno comprometido. Toda la aritmética se puede realizar en módulo 2^N donde $160 \leq N \leq 512$.[9]

1. El generador mantiene un estado interno Xi que varía constantemente.
2. El generador admite una entrada opcional Wi, se asumirá que es cero cuando no se produzca.
3. Cada salida se produce de la siguiente manera:
   a) $salida[i] = hash(W_i + X_i \ mod \ 2160)$
   b) $X_{i+1} = X_i + salida[i] + 1 \ mod \ 2160$

- **El generador RSAREF**

  Este PRNG se basa en dos operaciones: hash con MD5 y adición módulo 2^{128} con un diseño relativamente simple. Consiste en lo siguiente:[9]

  1. Un contador de 128 *bits* C_i
  2. Un método de procesar entradas X_i que hace lo siguiente
     $C_{i+1} = C_i + MD5(X_i) \ mod \ 2^{128}$
  3. Un método de producir salidas así:
     a) $salida[i] = MD5(C_i) \ mod \ 2^{128}$
     b) $C_{i+1} = C_{i+1} \ mod \ 2^{128}$

- **El generador Blum Blum Shub**

  Este generador que suele abreviarse BBS es el que actualmente tiene una prueba más sólida de fortaleza. Es también muy simple, se basa en residuos cuadráticos y su única desventaja es que requiere bastante cálculos en comparación con otros generadores clásicos.

  Se empieza eligiendo dos grandes primos p y q que al ser divididos entre 4 den resto 3. Sea n el producto de p por q. Se elige entonces un número aleatorio x primo con n. La semilla inicial para el generador y el método de calcular los siguientes valores de la sucesión son:[9]

[5] Sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional (NSA) de los Estados Unidos

1. $S_0 = x^2 \ mod \ n$
2. $S_{i+1} = S_i^2 \ mod \ n$

Deben emplearse sólo unos pocos *bits* del final de cada S_i. Siempre es seguro utilizar solamente el *bit* menos significativo. Si se usan no más de $log_2(log_2 \ S_i)$ de los *bits* de menos peso, se demuestra que predecir *bits* adicionales es tan difícil como la factorización de n, un problema tenido por intratable actualmente y base de métodos tan sólidos como RSA.

Otro detalle interesante de este generador es que se puede calcular directamente cualquier valor S_i sin tener que calcular los anteriores. En concreto:

$$S_i = (S_0^{2^i \ mod \ (p-1)(q-1)}) \pmod{n}$$

2.3. AUTÓMATAS CELULARES

Los Autómatas Celulares fueron introducidos a finales de los años 40 por John von Newmann siguiendo la sugerencia planteada en la Red Infinita de Stanislav Ulam con el objetivo de crear un modelo real del comportamiento de sistemas extensos y complejos.

Lo que hizo Ulam fué desarrollar a partir del fenómeno del crecimiento del cristal, un medio ambiente diferente: una red infinita, desplegada como un tablero de damas. Cada cuadrado de la red podría ser visto como una 'célula'. Cada célula sería una máquina separada de estados finitos que actuaría de acuerdo con un conjunto compartido de reglas.

Newmann por tanto replanteó su Autómata Autorreproductor, siendo este conocido como el primer Autómata Celular (AC). En un AC los objetos que pueden ser interpretados como datos y los que hacen la función de dispositivos de computación se encuentran al mismo nivel y sometidos a las mismas leyes, todo lo contrario a los modelos convencionales de computación tales como la máquina de Turing que hacen una distinción clara entre la estructura del computador y los datos sobre los que el computador trabaja.

El interés de Newmann estaba dirigido principalmente a dar una explicación a ciertos aspectos de la biología. Los mecanismos que propuso para la auto-replicación se asemejan bastante a los que utilizan los organismos vivos.

Tres décadas después con el "Juego de la vida" de John Conway los Autómatas Celulares alcanzaron gran popularidad. Se convirtió en un objeto de culto para una generación de jóvenes científicos cuando se publicó en la sección de juegos matemáticos

de la revista Scientific American en Octubre de 1970.[5]

- **Definición**

Es un sistema dinámico donde el espacio y el tiempo son discretos. Describe la evolución de un sistema espacialmente explícito. En cada instante de tiempo cada celda analiza el estado de un conjunto de celdas vecinas llamadas *vecindario*, su propio estado y, en función de un conjunto de reglas de evolución, se determina el cambio o no de estado de cada celda que constituye el espacio.[5]

- **CARACTERÍSTICAS**

Todo Autómata Celular tiene las siguientes características:

- El espacio está formado por un conjunto discreto de subespacios homogéneos, llamadas celdas distribuidas en una rejilla regular n-dimensional. Si el autómata es unidimensional las celdas se distribuyen a lo largo de una línea, si es bidimensional a través de una rejilla plana que puede ser triangular, cuadrada, hexagonal, etc.

- Cada celda puede estar en un único estado en determinado instante de tiempo. Este estado debe estar definido en un conjunto finito de estados asociados al espacio del autómata.

- Una configuración inicial que consiste en asignarle un estado inicial a cada celda del autómata.

- El estado de las celdas cambia de un instante a otro de acuerdo a un conjunto de reglas de evolución comunes a todas las celdas. Define como debe de cambiar de estado dependiendo del estado inmediatamente anterior de su vecindad. Con este conjunto de reglas podemos ver como partiendo de una situación inicial el mundo evoluciona y con sólo modificar levemente una de las reglas podemos observar como la evolución cambia drásticamente.

- Estas reglas son esencialmente una máquina de estado finito, especificadas en una tabla de reglas (también conocida como función de transición), con una entrada para todas las posibles configuraciones del vecindario.

- El vecindario de una celda está formado por las celdas adyacentes y se define igual para todas las celdas del autómata. Se define las reglas que indican si una célula es contigua a otra, indicando sus posiciones relativas respecto de la celda misma. Cuando el espacio es de tamaño finito siempre se considera condiciones de contorno periódicas, es decir, la celda en la posición 0 tiene como vecina a la celda de la posición N y viceversa.

- Un reloj virtual de Cómputo conectado a cada celda del autómata que generará pulsos que indican cuando debe establecerse los cambios de estado de las células según las Reglas de Evolución, haciendo que todas las células cambien de estado al mismo tiempo.

Podemos tener distintos casos entre una sucesión de estados: puede que tenga un único ancestro, que tenga más de un posible ancestro o que carezcan de ancestros.

- Organización: partiendo de un estado inicial el AC evoluciona reduciendo el número de configuraciones finales. Hay una disminución del desorden.

- Irreversibilidad local: en un AC diferentes situaciones iniciales pueden dar lugar a la misma situación final. De manera que conociendo sólo al hijo es imposible saber quien es el padre. No es posible volver atrás en el tiempo y reconstruir la historia completa.

- Jardines del Edén: Es el caso en que la sucesión de estados no tenga ancestros. Situaciones que sólo se dan como configuración inicial y no pueden presentarse como producto de la evolución del autómata, sino solamente al inicio de dicho proceso.[5]

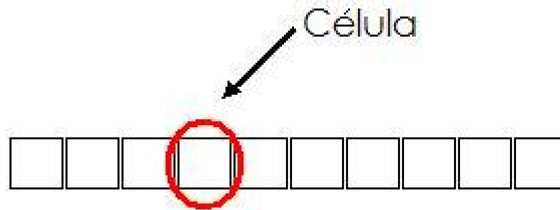
2.3.1. Autómatas celulares de Wolfram

Estructura

Un autómata celular de Wolfram consta de un arreglo lineal finito de celdas o células.(ver Fig. 9) Cada célula del arreglo puede tomar como valor un elemento de un conjunto finito de estados, el cual es denotado por la letra K. Los elementos del conjunto K pueden ser de diferentes tipos (números, letras, símbolos, etc.) debido a que la naturaleza de los estados no es importante. Para el procesamiento interno de las evoluciones del autómata celular lineal a través del tiempo lo más conveniente es utilizar números, esto para hacer más fáciles los cálculos, y para la representación final de las evoluciones es útil utilizar diferentes colores, para mejorar el aspecto visual. Aunque la

natulareza de los estados no influye en el comportamiento del autómata, la interacción que mantienen las células durante la evolución por medio de esos estados sí influye de manera directa en el comportamiento.

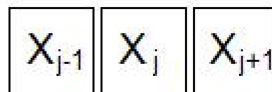
Figura 9: Arreglo de 10 Células



El número de estados para un autómata celular lineal puede ser variable, por esta razón en la notación Wolfram el número de estados se representa por medio de la variable k, es decir, la cardinalidad del conjunto K es igual a k, lo cual se escribe como $\#(K) = k$

Otro elemento importante en la notación Wolfram es el que él denomina como *radio de vecindad* y que denota por medio de la variable r . El radio de vecindad indica el rango de interacción a nivel local que van a tener las células entre sí. Si r es igual a uno, entonces cada célula del autómata cambiará su estado durante la evolución dependiendo de los estados de las células vecinas más cercanas, tanto del lado izquierdo como del lado derecho. Esto forma lo que se conoce como *vecindad*, donde el tamaño total de una vecindad es igual a $2r + 1$ incluyendo a la célula central. Para el caso en el que r es igual a uno, el tamaño total de la vecindad va a ser igual a 3 (los autómatas de este tipo se llaman *Autómatas Celulares de Wolfram*). (Fig. 10)

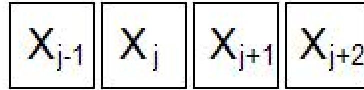
Figura 10: Vecindad de Tamaño 3



En este proyecto se usarán también autómatas celulares con $r = 1{,}5$ esto quiere decir que la evolución de una célula del autómata dependerá del estado de 1,5 células a lado izquierdo y derecho, sin embargo, como esto no es posible debido a que son indivisibles, las dos medias células que se tienen a lado y lado, se agruparán en una sola que se ubicará como indica la figura 11.

Se plantea ahora la pregunta: ¿Cuáles son los vecinos para la primera y última célula?. por ejemplo para la regla 105 que se muestra en la figura 12, si estas dos células no tuvieran vecinos el autómata no podría evolucionar infinitamente,(ver Fig. 13) la solución a este problema es muy sencilla, se debe definir una frontera periódica, de esta manera, el vecino del lado izquierdo para la primera célula es la última célula, y de

Figura 11: Vecindad de Tamaño 4



igual manera, el vecino del lado derecho para la última célula es la primera, como se muestra en la figura 14.

Figura 12: Regla 105

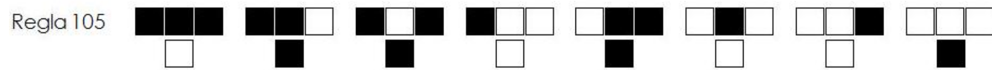
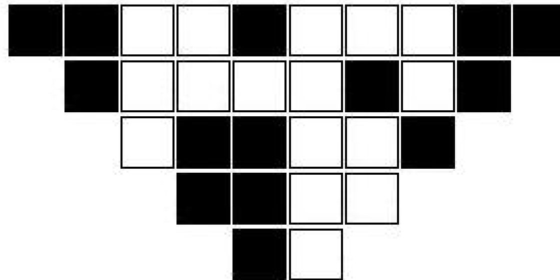


Figura 13: Evolución de la regla 105



- **Enumeración de los autómatas**

En este punto se ha hablado de las reglas 30 y 105, pero no se ha explicado como se asignan estos números, en realidad es muy sencillo, hay ocho posibles combinaciones binarias de tres números y los números del 0 al 255 se pueden escribir en 8 *bits*, lo que se hace es que el número escrito en binario correponda a la regla de evolución. Donde cada *bit* corresponde a una posible combiación de la vecindad.(ver Fig. 15)

2.3.2. Juego de la vida

Juego matemático inventado por John Horton Conway, un matemático de la Universidad de Cambridge, en 1970. Éste ocasionó en los años siguientes un gasto de millones de dólares en tiempo de computación desperdiciado: hubo hackers que se infiltraron en grandes computadores sólo para probar combinaciones de células.

Figura 14: Frontera periódica

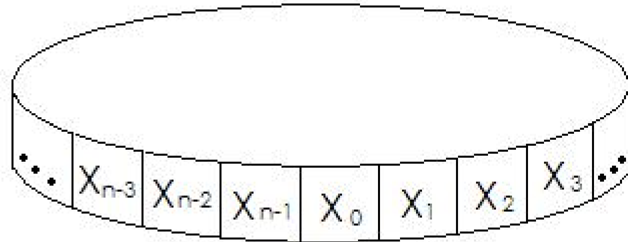


Figura 15: Enumeración de las reglas

	■■■	■■□	■□■	■□□	□■■	□■□	□□■	□□□
Regla 0	□	□	□	□	□	□	□	□
Regla 1	□	□	□	□	□	□	□	■
Regla 2	□	□	□	□	□	□	■	□
Regla 3	□	□	□	□	□	□	■	■
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Regla 252	■	■	■	■	■	■	□	□
Regla 253	■	■	■	■	■	■	□	■
Regla 254	■	■	■	■	■	■	■	□
Regla 255	■	■	■	■	■	■	■	■

“El Juego de la Vida” consiste en simular el comportamiento de organismos similares a colonias de virus. Se juega sobre una parrilla de extensión arbitraria (hay implementaciones del juego en se se hace que la parrilla sea cilíndrica, esférica o una cinta de Moebius).

La mejor manera de entender el juego es imaginarse un gran tablero uniforme. Llamaremos a cada celda del tablero una *célula* y al tablero completo el *espacio celular*. Cada célula es idéntica y conoce en cualquier instante el estado de sus vecinas (derecha, izquierda, arriba, abajo y las cuatro celdas en las diagonales). Una célula puede estar en uno de dos estados: viva o muerta. Con el paso del tiempo, la población de células cambia según ciertas reglas, dando paso a nuevas generaciones. Las reglas son las siguientes:

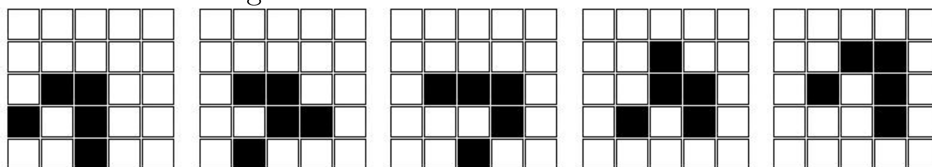
1. **Sobrevivencia**. Si una célula está rodeada por dos o tres células vivas en la presente generación, ésta permanecerá viva en la siguiente generación.

2. **Nacimiento**. Si una celda vacía (célula muerta) está rodeada en la actual generación por exactamente tres vecinos vivos, esta célula “nacerá” en la siguiente generación.

3. **Muerte por soledad**. Si una célula no tiene vecinos vivos, o tiene sólo una célula vecina viva, ésta morirá de soledad en la siguiente generación.

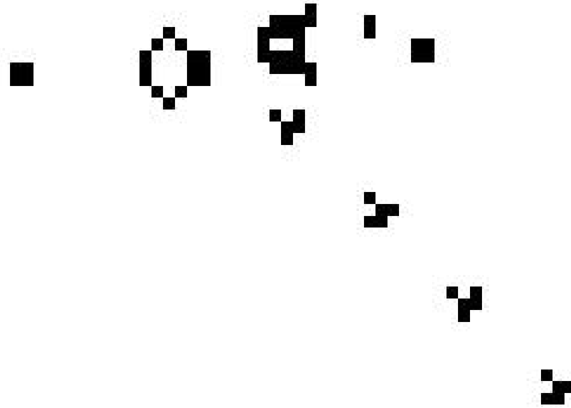
4. **Muerte por sobre-población**. Si una célula tiene cuatro o más vecinos vivos, ésta morirá en la siguiente generación por sobre-población.

Figura 16: Evolución de un deslizador



La figura 17 muestra un patrón llamado *Cañon de deslizadores*, éste con el paso del tiempo va generando deslizadores (*gliders* en inglés) que son patrones que se mueven por el espacio celular conservando su forma. (ver Fig. 16) Como estas dos figuras, existen infinidad de formas que tienen comportamientos tan complejos como generar números primos o incluso una que actúa como una sola célula pero a una mayor escala.

Figura 17: Cañon de deslizadores



Tomado de Wikipedia.org

2.4. AUTÓMATAS CELULARES COMO generadorES DE BITS PSEUDO ALEATORIOS

Cualquier autómata celular de dos estados puede considerarse como un mecanismo generador de *bits*. El procedimiento que consideraremos acá es sencillo, aunque no es el único. Dado un autómata celular n-lineal de Wolfram (un autómata lineal de tamaño n de dos estados con una vecindad de él mismo y los dos de al lado), con n impar, y dada una configuración inicial, iteramos *k - 1* veces el mismo, obteniendo k configuraciones.

Como sucesión de *bits* generada se puede considerar la sucesión temporal de la célula central, $\frac{(n-1)}{2}$.

De la misma forma, no habría ningún problema en considerar la sucesión temporal de cualquier otra célula como sucesión generada de *bits*. De acuerdo con esto, no hay problema en considerar el caso en que n sea par, eligiendo la evolución de una de las células centrales.

Así pues, sin más que elegir de forma aleatoria una determinada configuración inicial, que hace el papel de clave o semilla e iterar el AC determinado un número suficientemente grande de veces se obtendrá una sucesión de *bits*, de la cual se estudiarán sus propiedades de aleatoriedad.

2.5. LAS PRUEBAS ESTADÍSTICAS

En diferentes trabajos se recogen pruebas diseñadas para medir las propiedades de un generador de *bits* como generador de *bits* pseudo aleatorio. Es imposible dar una demostración matemática de que un generador es aleatorio [8], pero se pueden estudiar ciertas características que deben poseer.

Las pruebas, basadas en propiedades estadísticas, determinan si una sucesión posee ciertas características que una sucesión "realmente" aleatoria debería tener. La conclusión de cada prueba no es definitiva pero sí bastante probabilística.

Si una sucesión falla en alguna de las pruebas, el generador es rechazado al no ser pseudo aleatorio. Si una sucesión pasa todas las pruebas, su generador no es rechazado, pero no es prueba inequívoca de que genere sucesiones verdaderamente pseudo aleatorias.

2.5.1. Los Postulados de Golomb

Es difícil determinar, como ya hemos dicho, si una sucesión binaria es segura para su uso en Criptografía, es decir, si tiene realmente buenas propiedades aleatorias. Golomb formuló tres postulados que una sucesión binaria finita debe satisfacer para ser denominada pseudo aleatoria.

Para entender estos postulados es necesario tener algunos conceptos previos claros:

- Sea $s = \{s_0, s_1, s_2, ...\}$ una sucesión infinita. Una *subsucesión* de s de n términos se denota por $s^n = \{s_0, s_1, s_2, ..., s_{n-1}\}$

- Una sucesión s se dice que es *periódica de periodo N* (ó *N-periódica*) si $s_i = s_{i+N}$, para todo $i \geq 0$.

- Una sucesión s es *periódica* si es N-periódica para algún entero positivo. El *periodo* de una sucesión periódica es el menor entero positivo N para el cual es N-periódica.

- Si s es una sucesión periódica de periodo N, se denomina *ciclo* de s a la subsucesión s^N.

- Una *racha* de longitud k es una subsucesión de s de k dígitos iguales entre dos dígitos distintos. En una sucesión, si los dígitos iguales son ceros se denomina *hueco* o *gap*, y si son unos se denomina *bloque* o *block*.

- Sea s una sucesión de periodo N. La *función de autocorrelación* de s se define como sigue:

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

- La función $C(t)$ mide las coincidencias existentes entre la sucesión s y ella misma desplazada cíclicamente t posiciones. Si s es una sucesión aleatoria de periodo N, el producto $|N \cdot C(t)|$ debe ser muy pequeño para todos los valores de t, tales que $0 \leq t \leq N$.

Postulado 1. *En un ciclo s^N de s, el número de unos es aproximadamente igual al número de ceros. La diferencia entre uno y otro no debe exceder la unidad.*

Postulado 2. *En cada ciclo s^N, la mitad de las rachas del número total de rachas observadas tiene longitud 1, una cuarta parte longitud 2, una octava parte longitud 3, etc. Para cada una de estas longitudes debe haber el mismo número de bloques y huecos, o como mucho la diferencia entre uno y otro no debe exceder de la unidad.*

Postulado 3. *La autocorrelación C(t) toma dos valores. Así, para un entero k:*

$$N \cdot C(t) = \sum_{i=0}^{N-1} (2s_t - 1)(2s_{i+t} - 1) = \begin{cases} N & t = 0 \\ k & 1 \leq t \leq N-1 \end{cases}$$

Una sucesión finita que cumpla los tres postulados se denomina PN (Pseudo-Noise en inglés).

A continuación se muestran algunas de las prueba estadísticas que son utilizadas comúnmente para determinar si una sucesión binaria $s = \{s_0, s_1, s_2, ..., s_{N-1}\}$ posee determinadas características que una verdadera sucesión aleatoria debería tener (para más información ver [12] y [13]).

2.5.2. Prueba de Frecuencias (*Monobit test*)

Determina si el número de ceros y el número de unos en la sucesión s son aproximadamente los mismos. Si llamamos n_0 y n_1 al número de ceros y de unos de la sucesión, el valor estadístico a contrastar es:

$$X_1 = \frac{(n_0 - n_1)^2}{n},$$

que sigue una distribución χ^2 [6] con un grado de libertad ($l = 1$). Esta prueba se basa en el primer postulado de Golomb.

2.5.3. Prueba de Series (*Two-bit test*)

Busca y cuenta el número de ocurrencias de las cuatro subsucesiones: 00, 01, 10 y 11 y estudia si su distribución es tal y como se esperaría de una sucesión generada por un generador verdaderamente aleatorio. Se definen los parámetros n_{00}, n_{01}, n_{10} y n_{11} como el número de ocurrencias de cada una de las cuatro subsucesiones anteriores. Se verifica que $n_{00} + n_{01} + n_{10} + n_{11} = n - 1$.

El valor estadístico que se contrasta en esta prueba es:

$$X_2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

2.5.4. Prueba del Poker

Estudia todas las subsucesiones diferentes de un tamaño m que se determina previamente. El valor m es el mayor entero positivo que verifique que $[n/m] \geq 5 \cdot (2^m)$. Se toma entonces el valor $k = [n/m]$ y se divide la sucesión s en k partes no solapadas, cada una de ellas de longitud m. Se definen 2^m parámetros n_i para almacenar el número de ocurrencias de cada una de las sucesiones de la longitud indicada. Esta prueba determina si cada una de las posibles sucesiones de longitud m aparecen un número semejante de veces.

El valor estadístico estudiado es:

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} {n_i}^2\right) - k$$

que sigue una distribución χ^2 con $2^m - 1$ grados de libertad ($l = 2^m - 1$).

[6] Chi cuadrado

2.5.5. Prueba de rachas

La prueba toma una sucesión binaria s y calcula el número de subsucesiones de s formadas por una determinada cantidad consecutiva de ceros o de unos, y que no vienen ni precedidas ni seguidas por el mismo dígito. Si la racha está formada por ceros se conoce como *hueco* mientras que si está formada por unos se denomina *bloque*. Para determinar la longitud de las diferentes rachas a estudiar, primero se calculan los sucesivos parámetros e_i definidos según la siguiente expresión:

$$e_i = \frac{n-i-3}{2^{i+2}} \text{ para } 1 \leq i \leq k$$

donde k es el mayor valor de i que verifica que $e_i = 5$.

Se definen además los parámetros H_i y B_i que almacenan, respectivamente, el número de huecos y el número de bloques de longitud i. El valor estadístico a contrastar es:

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

que sigue una distribución χ^2 con *2k - 2* grados de libertad *(l = 2k – 2)*.

Las pruebas de series, rachas y póker se basan en el segundo postulado de Golomb.

2.5.6. Prueba de Autocorrelación

El propósito de esta prueba es analizar la correlación entre la sucesión s y la sucesión s' obtenida de desplazar sin rotación la sucesión s un número determinado de posiciones (d) a derecha o izquierda. El número de *bits* de s que no coinciden con la correspondiente sucesión desplazada d posiciones se denota por $A(d)$ y se calcula por la siguiente expresión:

$$A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}, \quad 1 \leq d \leq [n/2]$$

donde $\oplus$ es la operación "or exclusiva" (xor). El valor del estadístico a encontrar es:

$$X_5 = \frac{2 \cdot \left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$$

que sigue una distribución normal $N(0, 1)$.

Para valores pequeños de $A(d)$ se recomienda repetir la prueba para diferentes desplazamientos. Esta prueba se basa en el tercer postulado de Golomb.

2.5.7. Prueba estadística universal de Maurer

Otra de las pruebas ampliamente recomendada para analizar la aleatoriedad de una sucesión de *bits*, es la conocida como prueba universal de Maurer, que permite detectar algunas deficiencias en la supuesta aleatoriedad de un generador.

Para realizar la prueba de Maurer hay que calcular el valor estadístico para la sucesión de salida s. Se escoge un valor para un parámetro L dentro del intervalo $[6, 16]$. La sucesión se divide en bloques no superpuestos de L *bits*. El número total de bloques necesario es igual a $Q + K$, donde el mínimo valor para Q recomendado es $2^L \cdot 10$ y el mínimo valor de K recomendado es $2^L \cdot 1000$. En total, para el caso $L = 16$, hay que descomponer la serie en $2^L \cdot 1010$ (es decir, 66.191.360) bloques de 16 *bits* cada uno, lo que supone un total de 1.059.061.760 *bits*.

La principal desventaja de la prueba de Maurer es, por tanto, la enorme cantidad de *bits* (más de mil millones) que se requieren para poder someter el generador a estudio. La prueba obtiene un valor estadístico X_n que sigue una distribución Normal. A partir de la media esperada, de la desviación estándar calculada y del nivel de significación se obtienen los valores umbrales de aceptación o de rechazo k_1 y k_2. El valor estadístico debe quedar entre estos dos valores calculados.

El proceso se realiza para diferentes valores del parámetro L (entre 6 y 16). El valor del nivel de significación α que se recomienda esté entre 0,001 y 0,01. Para valores recomendados de L, al tomar el último extremo $L = 16$, se tiene que (tomando un valor de α =0,005) $k_1 - k_2 = 2 \cdot x_\alpha \cdot \sigma = 0{,}000853$. El valor estadístico calculado después de 65.536.000 sumas y logaritmos, debe caer en un intervalo de valores verdaderamente estrecho. (Tomado de [1])

2.5.8. Prueba Chi-cuadrado

Adicionalmente a las pruebas estadísticas anteriores podemos agregarle la prueba χ^2 (chi cuadrado) que se usa para determinar si una muestra de valores se aproxima a una distribución determinada. Para el presente proyecto se usará con el fin de verificar que el generador de números pseudo aleatorios propuesto se aproxima a una distribución uniforme.

La prueba Chi-cuadrado es un ejemplo de las denominadas pruebas de ajuste estadístico, cuyo objetivo es evaluar la bondad del ajuste de un conjunto de datos a una determinada distribución candidata. Su objetivo es aceptar o rechazar la siguiente hipótesis:

"Los datos de que se dispone son una muestra aleatoria de una distribución $F_X(x)$"

El procedimiento de realización de la prueba Chi-cuadrado es el siguiente:

1. Se divide el rango de valores que puede tomar la variable aleatoria de la distribución en K intervalos adyacentes:

   $$[a_0, a_1)\,, [a_1, a_2)\,, K, [a_{K-1}, a_K)$$

   Pueden ser $a_0 = -\infty$ y $a_K = \infty$.

2. Sea N_j el número de valores de los datos que tenemos que pertenecen al intervalo $[a_{j-1}, a_j)$.

3. 3) Se calcula la probabilidad de que la variable aleatoria de la distribución candidata $F_X(x)$ esté en el intervalo $[a_{j-1}, a_j)$. Por ejemplo, si se trata de una distribución continua, esa probabilidad sería:

   $$p_j = \int_{a_{j-1}}^{a_j} f_X(x)dx$$

   siendo $f_X(x)$ la función densidad de probabilidad de la distribución candidata. También se puede hacer:

   $$p_j = F_X(a_j) - F_X(a_{j-1})$$

   Nótese que este es un valor teórico, que se calcula de acuerdo a la distribución candidata y a los intervalos fijados.

4. Se forma el siguiente estadístico de prueba:

   $$\Delta = \sum_{j=1}^{K} \frac{(N_j - Np_j)^2}{Np_j}$$

   Si el ajuste es bueno, Δ tenderá a tomar valores pequeños. Rechazaremos la hipótesis de la distribución candidata si Δ toma valores "demasiado grandes".

   Nótese que para decidir si los valores son"demasiado grandes", necesitamos fijar un umbral. Para ello se hace uso de la siguiente propiedad.

   "Si el número de muestras es suficientemente grande, y la distribución candidata es la adecuada Δ tiende a tener a una distribución Chi-cuadrado de $(K-1)$ grados de libertad."

En realidad, la afirmación anterior sólo es estrictamente cierta si no hay que estimar ningún parámetro en la distribución candidata. Si para definir la distribución candidata hay que estimar algún parámetro (su media, su varianza,...) el número de grados de libertad de la distribución Chi-cuadrado es:

$$(K \text{ - } 1 \text{ - número de parámetros que hay que estimar})$$

Tenemos por tanto, que si la distribución candidata es la adecuada, conocemos la distribución del parámetro. Además, si la distribución candidata es la adecuada, el valor del parámetro tenderá a ser pequeño, y si no es adecuada, tenderá a ser grande.

Una forma razonable de fijar un umbral de decisión sería:

"Rechazar la distribución candidata si $\Delta > \chi^2_{gdl,\alpha}$ siendo $\chi^2_{gdl,\alpha}$ el valor que en la distribución Chi-cuadrado de *gdl* grados de libertad deja por encima una masa de probabilidad de α".

Es muy importante tener en cuenta que la prueba está sujeta a error. Acabamos de ver que es posible equivocarse aunque la hipótesis sobre la distribución candidata sea cierta, porque podemos tener la mala suerte de que los valores de Δ salgan grandes. Eso en todo caso sucederá con probabilidad baja (0.1, 0.05 ó 0.01, según acabamos de ver). Asimismo, podríamos equivocarnos también decidiendo que la distribución candidata es la adecuada aunque no sea cierto, debido a que los valores de Δ podrían salir pequeños. La prueba se basa en la suposición razonable de que si la distribución candidata no es la adecuada, los valores de tenderán a salir por encima del umbral $\chi^2_{gdl,\alpha}$.

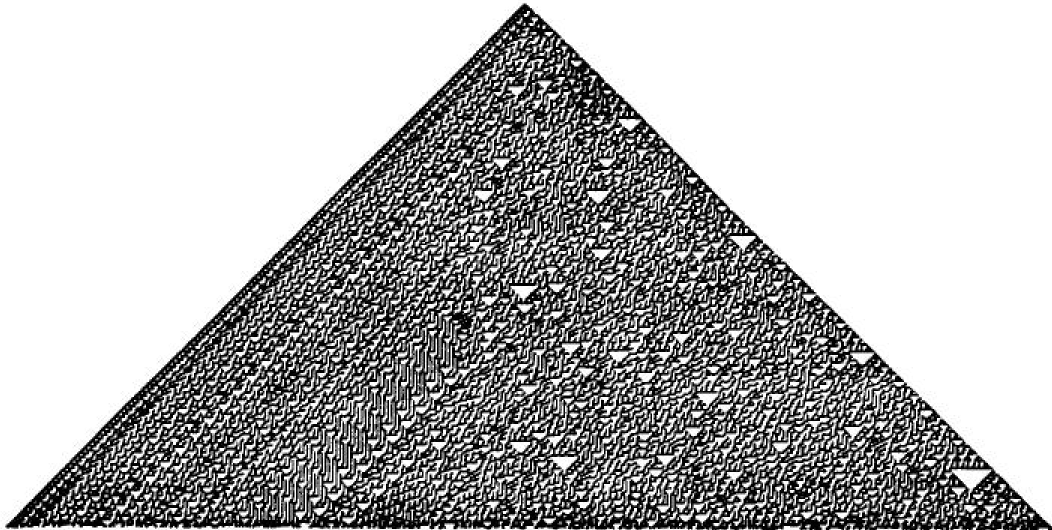
3. PROPUESTA

3.1. DESARROLLO DE LOS generadorES

Bastantes estudios se han realizado buscando la utilización de los autómatas celulares de Wolfram (aquellos de dos estados, radio 1 y vecindad 3) en criptografía. Los primeros estudios al respecto fueron realizados por él mismo en los años 80 y presentados a la comunidad científica en [15], ahí se propone el uso de la regla 30 como fuente de sucesiones pseudo aleatorias debido a que la evolución de ésta genera patrones bastante complejos (Fig. 18).

Sin embargo pocos estudios se han realizado con respecto a la utilización de de autómatas de vecindad 4 en este campo y por eso el presente proyecto se enfocará básicamente en estos. Para generar las sucesiones pseudo aleatorias, en principio, se hará de la manera propuesta por Wolfram pero adaptándola a una vencindad 4, este enfoque como lo vimos en la sección **2.2.1**, presenta características especiales que se muestran en la figura 11. Vale la pena anotar que así como los autómatas de Wolfram tienen un total de 256 posibilidades (2^{2^3}) para el caso de vecindad 4 existen un total de 65536 autómatas (2^{2^4}).

Figura 18: Evolución de la regla 30



Tomado de [15]

3.1.1. Metodología de las pruebas

Como se explicó anteriormente son cinco pruebas diseñadas para comprobar los tres postulados de Golomb y por tanto poder considerar una sucesión como criptográficamente segura. Sin embargo no basta con que una sola sucesión pase las cinco pruebas para decir que el generador que la produjo es criptográficamente seguro, las pruebas deben hacerse con una cantidad suficiente de cadenas de tal manera que un buen porcentaje las pasen.

Las pruebas se harán de la siguiente manera: para cada uno de los 65536 autómatas se generarán 100 cadenas de 20000 *bits* cada una. (recomendado en [12]) Un generador se considerará criptográficamente seguro si al menos 90 cadenas pasan cada una de las cinco pruebas. El orden en que se realizan las pruebas es el siguiente: prueba de frecuencias (P_1), prueba de series (P_2), prueba de rachas (P_3), prueba del póker (P_4) y prueba de autocorrelación(P_5). Dado una autómata determinado, únicamente se le aplicará la prueba autocorrelación si ha pasado, con el margen definido, todas las otras pruebas estadísticas, es decir, en el momento en que falle alguna de las pruebas, las demás no se hacen.

Los autómatas que pasaron las pruebas se muestran en el cuadro 1

# AC	P_1	P_2	P_3	P_4	P_5	# AC	P_1	P_2	P_3	P_4	P_5
2805	93	92	90	91	95	27242	98	99	96	91	98
4590	95	97	94	91	95	27285	97	97	99	95	96
5610	92	92	93	94	97	27289	93	95	91	97	97
6630	92	91	91	90	94	27290	93	94	99	98	100
7710	96	97	94	92	97	27301	94	96	98	93	97
7905	96	97	97	95	93	27305	92	93	96	92	92
8925	98	99	92	93	93	27795	94	94	91	92	93
9690	95	91	94	90	93	28050	94	97	91	91	94
9945	93	94	90	92	96	28305	92	91	94	93	93
10710	93	90	90	93	98	30345	92	92	92	90	97
11565	97	98	91	96	97	30840	95	97	95	97	92
11730	95	93	91	94	94	30855	95	99	92	96	97
15420	99	100	97	91	91	31110	94	92	94	93	95
17340	93	92	90	90	95	31738	94	98	96	100	100
17595	90	92	95	94	95	34425	94	94	93	93	97
18360	94	96	92	92	99	34680	95	97	94	93	95
18870	96	94	95	93	97	34695	90	93	92	90	96
19125	90	93	97	93	99	35190	95	96	92	94	96
19275	99	98	98	98	92	37485	94	97	91	92	91
21165	92	95	96	92	98	37740	94	93	92	91	97
21846	98	99	97	94	94	37995	96	92	93	92	95
21861	98	97	97	96	92	38229	95	94	94	94	95

# AC	P_1	P_2	P_3	P_4	P_5	# AC	P_1	P_2	P_3	P_4	P_5
21862	94	96	94	90	98	38230	94	95	96	90	97
21865	92	95	97	96	97	38233	94	98	95	95	96
21866	97	96	94	95	91	38245	96	91	93	91	90
21910	98	98	93	96	93	38246	96	94	94	92	96
21913	95	94	94	90	93	38249	91	94	96	91	94
21914	96	97	96	98	90	38250	95	97	93	94	97
21925	96	96	99	97	99	38293	94	93	92	93	92
21926	98	97	90	96	98	38294	93	91	97	96	94
21929	95	92	97	95	97	38297	95	93	96	94	96
22101	94	98	94	96	92	38309	94	92	97	93	92
22102	95	95	96	93	98	38310	97	95	96	93	94
22105	94	96	97	93	98	38313	94	90	93	91	96
22117	97	97	97	96	95	38485	97	95	94	92	93
22118	96	97	95	93	97	38489	97	98	95	98	97
22121	94	91	95	90	97	38490	94	93	97	94	93
22122	92	93	94	94	91	38501	96	99	94	96	96
22165	96	93	96	94	97	38506	96	97	97	92	94
22166	94	97	94	94	91	38549	97	97	98	93	95
22170	96	98	98	95	93	38550	94	97	97	90	93
22181	95	97	95	94	92	38553	97	96	97	91	93
22185	99	95	97	92	93	38554	93	96	95	94	94
22186	95	96	95	97	96	38565	94	97	94	96	93
22695	96	96	94	92	97	38566	99	95	95	92	98
22869	95	96	93	96	94	38569	96	91	97	93	94
22870	99	99	97	93	93	38570	97	96	91	94	98
22873	95	97	94	94	97	39015	95	93	93	93	97
22874	90	93	92	91	93	39253	96	96	98	94	93
22889	93	96	94	92	98	39254	97	98	91	99	95
22890	95	97	94	96	97	39257	99	98	97	96	100
22933	93	94	93	90	96	39258	93	93	94	90	94
22934	100	98	95	92	96	39269	95	94	99	97	97
22937	99	100	96	96	97	39270	97	97	94	95	90
22938	96	94	97	95	94	39273	96	94	96	90	97
22949	97	94	94	91	95	39274	91	91	93	93	99
22950	95	96	93	99	97	39317	94	95	96	95	95
22953	95	95	94	92	93	39322	92	94	92	97	97
22954	95	95	97	93	90	39333	91	94	99	93	92
23125	97	98	94	94	94	39334	94	92	92	94	97
23126	96	97	94	95	98	39337	95	95	97	94	94
23129	93	94	97	92	91	39510	96	96	94	94	95
23141	98	96	98	95	96	39513	94	94	97	90	96
23142	100	100	99	91	96	39514	92	91	94	96	97

# AC	P_1	P_2	P_3	P_4	P_5	# AC	P_1	P_2	P_3	P_4	P_5
23145	96	96	95	91	94	39526	98	96	97	97	97
23146	91	90	91	94	95	39529	94	94	97	97	95
23160	93	93	93	94	95	39573	97	97	96	96	97
23189	98	92	93	94	94	39574	97	94	99	95	95
23190	96	94	96	95	99	39589	96	98	96	96	95
23193	97	96	97	93	91	39593	92	97	92	99	95
23194	98	95	97	95	93	42325	94	96	94	94	98
23206	98	99	90	96	95	42326	96	92	96	94	92
23209	91	93	94	94	94	42329	99	97	91	92	98
25245	94	94	92	90	96	42341	90	94	96	91	93
25500	95	91	93	90	94	42345	90	93	97	91	93
25941	96	99	94	91	96	42346	93	96	94	90	96
25942	93	93	92	92	95	42389	95	96	94	93	92
25946	96	92	97	92	95	42390	95	96	96	92	96
25957	99	98	93	94	97	42393	98	99	94	93	94
25961	92	97	97	95	96	42394	94	94	95	95	93
25962	95	95	95	95	95	42406	95	96	97	93	94
26005	98	98	95	96	97	42409	96	96	96	99	92
26006	97	98	94	93	94	42585	94	94	98	95	93
26009	96	97	92	98	93	42586	97	96	95	92	94
26010	93	93	96	95	95	42597	96	94	97	95	98
26021	95	97	92	96	98	42601	93	96	94	95	96
26022	94	94	97	90	96	42645	98	95	94	91	98
26025	91	96	98	94	93	42646	90	92	94	95	96
26197	96	93	93	93	96	42649	93	90	95	94	99
26198	93	92	92	96	94	42650	91	96	92	95	92
26201	96	95	97	95	96	42661	97	95	96	94	99
26202	94	92	96	91	94	42665	97	96	95	93	98
26213	99	99	98	96	98	43349	96	98	97	96	93
26217	93	95	95	95	94	43353	97	100	100	95	92
26218	94	92	97	92	94	43354	93	93	94	94	96
26261	94	96	96	96	96	43365	98	98	93	93	99
26262	92	90	95	93	95	43366	94	93	96	92	96
26265	92	93	94	92	92	43369	96	94	97	98	95
26266	93	94	94	93	90	43370	99	94	96	93	95
26277	96	97	95	97	97	43413	92	93	94	94	91
26281	96	97	93	93	98	43414	97	98	96	94	95
26282	92	93	95	94	99	43418	97	95	91	97	98
26520	92	91	90	92	94	43429	93	97	96	96	95
26965	95	93	90	93	93	43433	99	98	98	94	95
26966	97	97	96	97	97	43606	98	96	92	90	99
26969	96	96	93	95	96	43609	96	98	93	92	96

# AC	P_1	P_2	P_3	P_4	P_5	# AC	P_1	P_2	P_3	P_4	P_5
26970	94	95	95	91	97	43625	98	94	95	96	97
26981	96	97	94	92	100	43669	98	95	95	93	97
26982	93	90	97	96	98	43670	95	91	94	92	95
26985	93	94	97	93	96	43673	98	99	90	94	100
27034	93	94	99	94	94	44370	93	96	96	95	93
27046	96	100	93	95	90	44625	93	93	91	92	99
27049	98	96	95	94	97	46155	98	97	94	91	95
27221	94	98	97	96	95	46410	99	98	95	94	93
27222	94	90	93	91	95	46665	91	90	96	93	93
27225	93	96	98	95	98	50115	99	100	97	91	96
27226	94	95	95	92	95	50235	92	90	91	90	93
27237	97	96	96	96	97	57630	92	96	94	90	91
27238	93	96	96	94	94	58905	95	91	93	92	95

Cuadro 1: Autómatas que pasaron las pruebas estadísticas

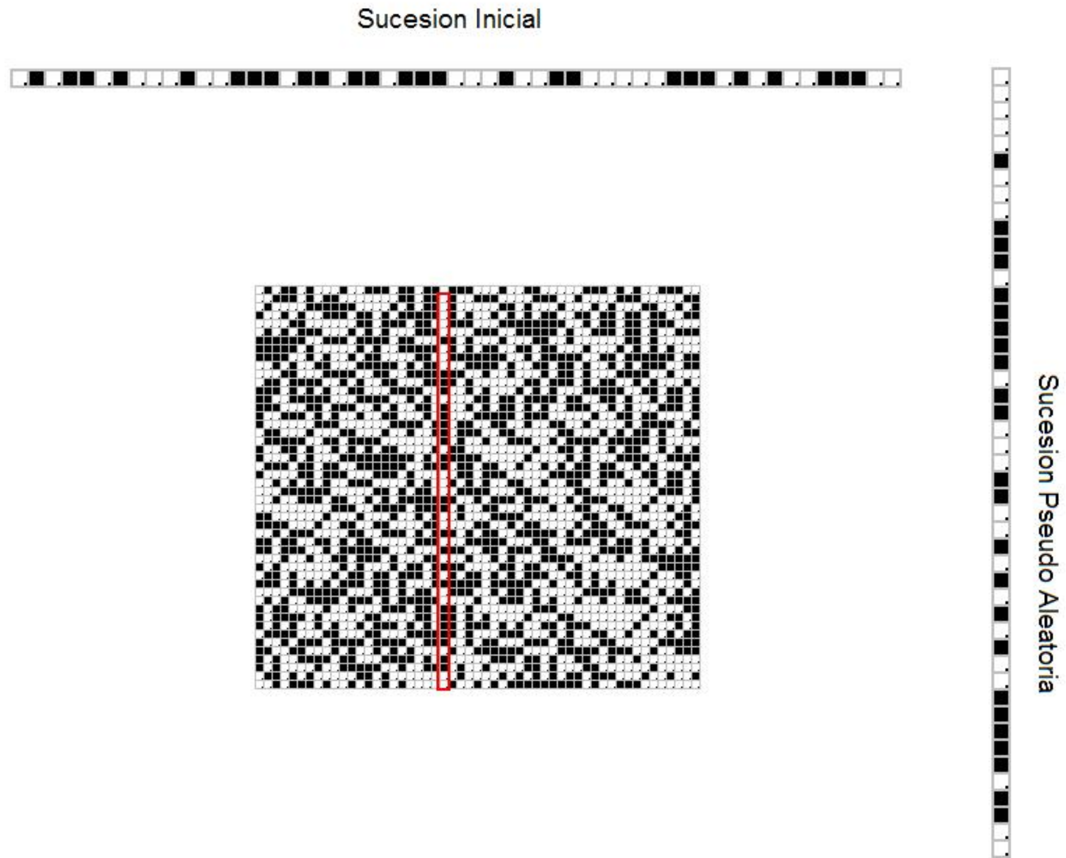
3.1.2. Método 1

Ahora se tiene una primera lista de autómatas que generan sucesiones pseudo aleatorias y con ellos es posible hacer la primera implementación de un algoritmo encriptador/desencriptador. Se denominará Método 1:

Explicación del algoritmo para el **Método 1**:

1. Se genera la sucesión cifrante a partir de la semilla (sucesión inicial). La longitud de la sucesión cifrante vendrá dada por la longitud del mensaje a encriptar. Ésto se hace de la siguiente manera:
   a) Se inicializa el estado del autómata con el valor de la semilla.
   b) Se evoluciona el estado actual del autómata.
   c) Se escoge como *bit* de la sucesión cifrante el estado de la célula central.
   d) Se actualiza el estado del autómata.
   e) Se repite el proceso desde el literal *b*) hasta tener una sucesión del tamaño del mensaje a encriptar.
2. Se escribe el mensaje a cifrar como una sucesión de *bits* utilizando para esto la representación ASCII de los caracteres.
3. Se aplica la función *xor bit* a *bit* entre el mensaje a encriptar y la sucesión cifrante. La salida de este procedimiento es el mensaje encriptado.

Figura 19: Método 1. Paso 1.



El proceso de desencriptado es similar dado que nuestro algoritmo es simétrico. El receptor del mensaje, o simplemente la persona que quiera ver el contenido en texto claro tiene que digitar la clave para generar la sucesión cifrante. Como la función *xor* es una involución cuando el mensaje encriptado es "*xoreado*" con la sucesión cifrante, el resultado es el mensaje en texto claro.

Supongamos que queremos encriptar las palabras "MENSAJE SECRETO" con la clave "12345678". La representación en binario del mensaje es:

1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0
1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0
1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 0 1 0

la sucesión cifrante generada por el autómata 30840 es:

1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 0 1
0 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1
1 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0

después de aplicar la función *xor* el resultado en binario es:

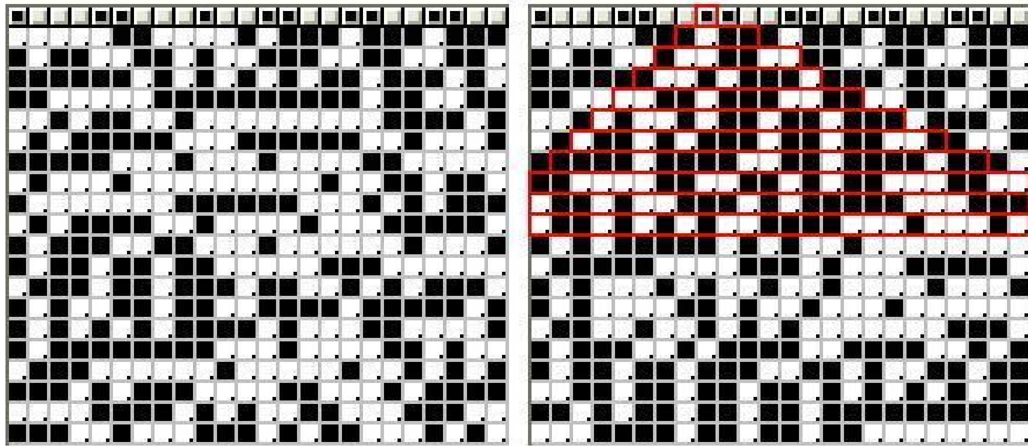
0 0 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1
0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 1 0 1 0

con lo que el mensaje a enviar en codigo ASCII sería:

r ò € à ; E Î \ 2 _ - » › Đ □

En la figura 20 se muestra la propagación del cambio de un solo *bit* en la semilla. Se puede observar que con solo unas cuantas evoluciones los estados del autómata varían considerablemente. Si por ejemplo un atacante conoce parte de la clave, ésta le sería inútil si se hiciera, por asi decirlo, una inicialización del autómata, de tal manera que la sucesión cifrante se escoja a partir del punto en el que todos los *bits* de la semilla influyan directamente en los estados de las células elegidas.

Figura 20: Propagación del cambio de un *bit*



Como se ve en la figura 20, para la primera evolución, de un *bit* de la semilla dependen directamente cuatro *bits* de la evolución, de estos cuatro *bits* dependen directamente siete *bits* de la siguiente evolución, y de estos a su vez, dependen diez *bits* del siguiente estado. En general después de n evoluciones, de un *bit* de la semilla dependerán directamente $1 + 3n$ *bits* del estado actual. De esto se puede deducir que si se tiene una semilla de l *bits* de longitud se deberán hacer $n = \frac{l-1}{3}$ evoluciones para que todos los *bits* del estado dependan directamente de todas las células de la semilla.

3.1.3. Método 2

El método 1 tiene varias deficiencias, la clave, no puede ser usada más de una vez pues el algoritmo perdería toda su seguridad y sería altamente vulnerable al ataque de clave reutilizada, además la clave debe ser acordada personalmente o se debe contar con un sistema de intercabio de claves que sea seguro. Una manera de solucionar el primer problema la propone Moad Benkiniouar y Mohamed Benmohamed en su artículo *Cellular automata for Cryptography* [3] y consiste en concatenar la clave del usuario con otra clave llamada *nonce* que es generada aleatoriamente. Esta clave deberá ser, después de encriptado el mensaje, añadida como texto claro al mensaje a enviar. Esto para que al momento de la desencriptación se cuente con las dos partes de la clave usada durante la encriptación. Sin embargo se debe tener en cuenta que el *nonce* deber ser lo suficientemente largo como para que la probabilidad de que se repita una pareja de claves sea muy baja. Si por alguna razón un mensaje es encriptado con la misma pareja (clave secreta / *nonce*) el sistema podrá ser violado.

Tomando la idea del *nonce*, se sugirió lo siguiente:

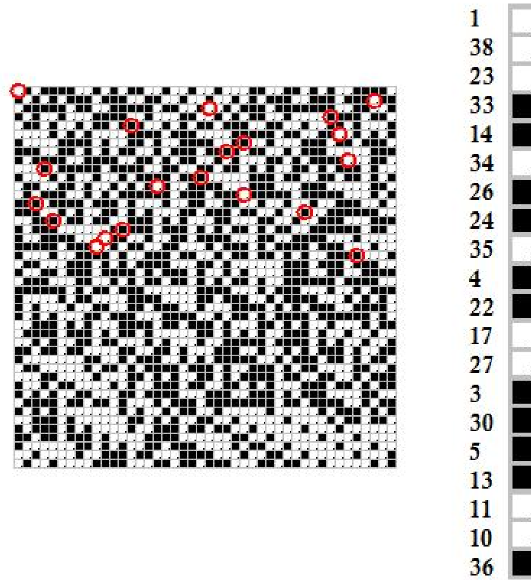
Diseñar un generador de números pseudo aleatorios (ya no sucesiones de *bits*) con el fin de no solo escoger una columna del autómata sino hacer posible una elección al azar de la célula que va a formar parte de la sucesión cifrante. La idea se refleja en la figura 21, donde los números que se muestran en la columna corresponden a salidas del generador y las células al lado de estos corresponden a las escogidas durante la evolución del autómata.

El algoritmo propuesto para el generador pseudo aleatorio es el siguiente:

1. Se genera un número binario de n *bits* a partir del *nonce* (sucesión inicial).
    a) Se inicializa el estado del autómata con el valor del *nonce*.
    b) Se evoluciona el estado actual del autómata.
    c) Se escoge como *bit* del número binario el estado de la célula correspondiente a la célula central.
    d) Se actualiza el estado del autómata.
    e) Se repite el proceso desde el literal *b*) hasta tener el número binario de n *bits*.
2. Se obtiene la representación decimal del número binario
3. Se divide este número entre $2^{n+1} - 1$ para tener un número tal que $0 \leq x \leq 1$

Sin embargo no solo es suficiente con tener el algoritmo hecho, se debe probar que el generador cuenta con las características de uniformidad y correlación necesarias para

Figura 21: Selección Aleatoria de Células



que sea considerado como un buen generador pseudo aleatorio.

Para determinar estas características se optó por hacer una prueba chi cuadrado (contra una distribución uniforme) a todos los autómatas que habían pasado las primeras cinco pruebas estadísticas. Se generaron 1′000,000 números, se dividieron en 100 clases y se tomó un nivel de significancia de 5 %.

Los autómatas que pasaron la prueba fueron los siguientes:

# AC	χ^2	# AC	χ^2	# AC	χ^2	# AC	χ^2
4590	81.6535	23141	67.0963	27305	105.1246	27301	81.2211
6630	122.0491	23142	101.0847	28050	113.6353	39526	97.5303
7710	98.1973	23145	99.7774	30840	112.7824	39573	83.7427
7905	92.2819	23146	98.4011	30855	107.6406	39574	93.6664
9690	109.2279	23190	104.7082	31110	99.4627	39589	95.145
10710	98.1414	23193	83.0775	34425	82.8752	39593	118.4494
11730	77.8824	23194	86.9938	34680	102.8867	42325	105.2119
15420	110.0045	23206	86.4256	34695	96.5662	42326	96.8957
17595	108.8133	23209	81.6267	37485	96.2729	42329	111.419
18360	115.9684	25941	81.5747	38229	110.2729	42341	80.607
19275	86.4139	25942	106.5031	38230	104.3591	42345	99.4188
21846	111.013	25946	101.0015	38233	93.4066	42346	102.4061
21861	110.0913	25957	98.9194	38245	118.9644	42389	119.5606

# AC	χ^2	# AC	χ^2	# AC	χ^2	# AC	χ^2
21862	102.2125	25961	105.6149	38246	103.7959	42390	100.8159
21865	83.0422	25962	111.7366	38249	98.3809	42393	78.4543
21866	100.9228	26005	103.4089	38250	81.7191	42394	72.3356
21910	108.1989	26006	119.2386	38293	90.9762	42406	96.2616
21913	72.6582	26009	113.3522	38294	86.955	42409	93.7792
21914	73.6021	26021	100.7605	38297	93.6642	42585	89.4517
21925	105.1118	26022	103.492	38309	109.1997	42586	106.8967
21926	108.7262	26025	101.4649	38310	98.0892	42597	108.7836
21929	115.2376	26198	114.9726	38313	112.4501	42601	110.3546
22101	117.6628	26201	74.4489	38485	109.6703	42645	100.8772
22102	106.9943	26202	108.8224	38489	78.9743	42646	115.3728
22105	76.86	26213	82.1123	38490	100.0791	42649	103.5912
22117	95.8428	26217	111.5646	38501	77.034	42650	88.6771
22118	94.5191	26261	113.278	38506	107.9361	42661	88.3602
22121	116.0008	26262	90.2086	38550	95.0899	42665	88.2037
22122	104.6936	26265	78.2016	38553	95.812	43349	118.0939
22165	108.5798	26277	94.4156	38554	92.8262	43353	91.4383
22166	88.784	26281	78.5636	38565	91.8508	43354	117.8047
22170	107.7124	26282	98.6937	38566	75.8859	43366	122.7695
22181	91.721	26965	88.0591	38569	112.8198	43369	100.9582
22185	95.2333	26966	100.0776	38570	83.8142	43370	117.4593
22186	83.4075	26969	114.0854	39015	101.7353	43413	105.914
22695	109.4323	26970	102.104	39253	106.1473	43414	119.2968
22869	73.318	26981	90.9896	39254	95.6863	43418	94.4231
22870	98.788	26982	90.8663	39257	87.7984	43429	66.0661
22873	82.6374	26985	98.4416	39258	103.9071	43433	111.9981
22889	82.5547	27034	89.0228	39269	96.6987	43606	113.437
22933	119.8145	27046	100.4532	39270	101.9509	43609	113.5674
22934	101.1696	27049	84.3915	39273	87.4964	43625	103.0993
22937	83.6227	27221	101.0232	39274	103.7068	43669	101.4537
22938	107.7749	27222	111.6319	39317	92.869	43670	90.6851
22949	69.6923	27225	83.8153	39322	122.0193	43673	66.0652
22950	90.843	27226	92.0311	39333	72.2325	44370	119.9822
22953	114.2475	27241	108.1693	39334	94.7973	46155	81.5031
22954	92.316	27242	112.4745	39337	84.3411	46665	112.4397
23125	100.0095	27285	95.8996	39510	107.4565	50115	110.877
23126	93.1856	27289	108.0145	39513	104.6373	57630	82.3604
23129	102.7977	27290	93.2205	39514	89.1975		

Cuadro 2: Autómatas que pasaron la prueba Chi Cuadrado

Explicación del algoritmo para el **Método 2**

1. Se genera la sucesión cifrante a partir de la semilla (sucesión inicial).
   a) Se inicializa el estado del autómata con el valor de la semilla.
   b) Se genera un número pseudo aleatorio entre 1 y el tamaño de la semilla.
   c) Se evoluciona el estado actual del autómata.
   d) Se escoge como *bit* de la sucesión cifrante el estado de la célula correspondiente al número generado.
   e) Se actualiza el estado del autómata.
   f) Se repite el proceso desde el literal *b*) hasta tener una sucesión del tamaño del mensaje a encriptar.
2. Se escribe el mensaje a cifrar como una sucesión de *bits* utilizando para esto la representación ASCII de los caracteres.
3. Se aplica la función *xor bit* a *bit* entre el mensaje a encriptar y la sucesión cifrante. La salida de este procedimiento es el mensaje encriptado.

Con el algoritmo del método 2 listo se debe ahora probar que las cadenas generadas cumplen con las condiciones de seguridad criptográfica necesarias. Para esto se hará uso de las pruebas estadísticas presentadas en la sección **2.4**.

Para este caso no se tendrán en cuenta los autómatas que no pasaron las primeras pruebas debido a que las cadenas generadas con estos no son criptográficamente seguras. Además se deben probar cada uno de los autómatas que generan números uniformes con todos los autómatas que habían pasado anteriormente. Por la cantidad de resultados obtenidos (mas o menos 400 paginas) se debió subir el margen con el que se pasaban las pruebas hasta 96. Co esto se consiguió reducir el número de páginas hasta 10. Los resultados se muestran en el Anexo A.

Supongamos que queremos encriptar las palabras "Mensaje Secreto" con la clave "12345678". La representación en binario del mensaje es:

1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0
1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0
1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 0 1 0

la sucesión cifrante generada por los autómatas 27221 en el generador de numeros pseudo aleatorios y 26970 como autómata principal es:

1 0 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1
0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0
0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0

después de aplicar la función *xor* el resultado en binario es:

0 0 1 0 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1
1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 0
1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0

después de aplicar la función *xor* el resultado en ASCII es:

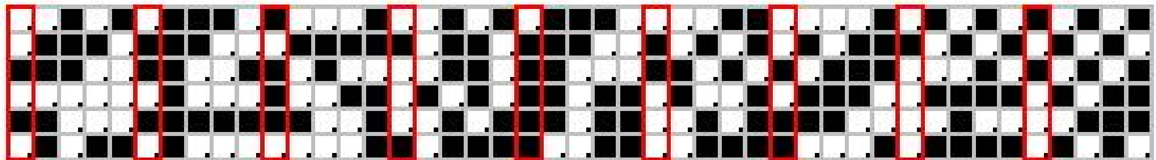
´ / œ > } ê ä “ ½ Z t À T L

3.1.4. Incremento en la velocidad de generación de la sucesión cifrante

En el método uno por cada evolución del autómata se genera un *bit* de secuencia cifrante, sin embargo, para el método dos, hay dos autómatas y para generar un solo *bit* es necesario hacer 13 evoluciones. Si el generador pseudo aleatorio devolviera más de un número por cada ejecución, el tiempo de generación podría disminuirse en gran medida. Por ejemplo, si en vez de tomar una sola columna del autómata, se tomarán 32, separadas espacialmente de tal manera que ninguna de las columnas intervenga directamente sobre las otras, (ver figura 22) el tiempo de ejecución sería casi igual al del método uno puesto que el número de evoluciones necesarias para generar un *bit* de sucesión serían: 1 evolución del autómata principal más 12/32 evoluciones del generador pseudo aleatorio.

Además si por cada evolución del autómata principal se tomarán más de un *bit* (dependiendo de las posiciones dadas por el generador pseudo aleatorio) la velocidad podría ser aun mejor. Para ser más claro, el generador pseudo aleatorio devuelve 32 posiciones para escoger *bits* del autómata principal, si escojo estas 32 posiciones de la misma evolución entonces el número de evoluciones necesarias para generar un *bit* de sucesión cifrante serían: 1/32 evoluciones del autómata principal y 12/32 evoluciones del generador pseudo aleatorio que es igual a 13/32 evoluciones por *bit* que es menos de la mitad de evoluciones necesarias que con el método 1.

Figura 22: Celdas elegidas del generador



El algoritmo propuesto para mejorar el generador pseudo aleatorio es el siguiente:

1. Se inicializa el estado del autómata generador con el valor del *nonce*.

2. Se evoluciona n veces el autómata generador escogiendo 32 columnas. simultáneamente.

3. Se encuentra la representación decimal de los números binarios generados.

4. Se divide esta representación entre $2^{n+1} - 1$ para obtener un número x tal que $0 \leq x < 1$.

A continuación se le aplicará la prueba Chi Cuadrado al algoritmo del generador pseudo aleatorio mejorado. Los resulados son los siguientes:

# AC	χ^2	# AC	χ^2	# AC	χ^2	# AC	χ^2
4590	113.6516	23145	97.3474	27241	111.08	39526	122.6988
8925	100.2892	23146	84.3544	27285	118.505	39529	101.0996
10710	110.126	23189	91.9772	27289	104.7814	42325	107.5088
11730	98.3372	23193	74.9798	27301	91.0366	42326	89.4368
15420	97.7812	23209	103.667	27305	90.7778	42329	74.6218
17595	88.3022	25500	96.7766	27795	81.0676	42341	116.9392
18870	92.2266	25941	84.7548	28050	106.7382	42346	95.0806
21861	97.321	25942	114.9536	34425	104.5742	42389	109.748
21862	84.3688	25946	104.2786	34680	121.4222	42390	101.556
21865	122.758	25961	120.304	35190	112.9778	42393	121.037
21866	108.026	26005	104.5454	37485	88.684	42394	111.5178
21913	97.9746	26006	104.599	37740	98.4396	42406	91.8412
21914	97.5746	26009	81.7926	37995	96.4106	42585	85.9412
21925	87.2762	26010	117.1156	38233	113.6606	42586	93.5058
21926	89.3002	26025	89.3344	38245	91.6398	42597	112.7186
21929	111.8464	26197	93.5218	38246	120.4496	42601	89.8076
22105	102.1986	26198	95.4866	38249	109.9714	42645	85.7616
22117	92.6016	26201	116.1408	38250	108.4388	43349	110.9826
22121	98.2316	26202	101.4048	38297	115.2426	43353	86.364
22122	118.2734	26213	95.808	38309	86.4964	43354	114.5874
22170	102.9162	26217	95.343	38310	76.9286	43365	116.1758
22181	108.0762	26218	91.5414	38489	86.2972	43366	110.1388
22185	92.9054	26261	99.6698	38490	98.814	43369	91.7436
22869	90.9788	26262	90.9628	38501	104.3552	43370	89.6626
22870	103.335	26265	93.5002	38506	101.5806	43413	119.1142
22874	87.901	26266	89.9274	38553	89.035	43414	90.2554
22889	113.3864	26277	106.1552	38565	121.5478	43429	98.2114
22890	110.714	26520	86.653	39015	114.2226	43609	105.7172
22934	109.3438	26965	92.6168	39253	79.4466	43625	87.748
22937	98.4944	26966	84.1582	39254	83.5328	43669	109.7512
22938	112.1336	26981	95.2614	39258	97.0278	44370	107.0506
22949	108.2656	26982	99.9708	39269	114.51	46155	76.8134

# AC	χ^2	# AC	χ^2	# AC	χ^2	# AC	χ^2
22950	77.2468	27034	78.2746	39270	88.487	46410	107.4574
22953	84.5346	27046	82.8916	39273	102.156	46665	86.6822
22954	98.6532	27049	92.552	39322	87.558	50115	97.7922
23125	93.2268	27221	86.9942	39333	94.0892	57630	110.5476
23126	80.3084	27225	100.5136	39334	101.995	58905	94.5628
23142	107.3772	27237	113.294	39514	86.8174		

Cuadro 3: Autómatas que pasaron la prueba Chi Cuadrado con el algoritmo mejorado

3.1.5. Método 3

Ahora se debe adaptar el algoritmo del **Método 2** para que se seleccionen varias células de cada evolución como se propone anteriormente. Al siguiente algoritmo se le llamará **Método 3**.

Explicación del algoritmo para el **Método 3**

1. Se genera la sucesión cifrante a partir de la semilla (sucesión inicial).
    a) Se inicializa el estado del autómata con el valor de la semilla.
    b) Se generan los 32 números pseudo aleatorios entre 1 y el tamaño de la semilla.
    c) Se evoluciona el estado actual del autómata.
    d) Se multiplican los valores pseudo aleatorios generados por la longitud de la semilla.
    e) Se escogen como *bits* de la sucesión cifrante los estados de las células correspondientes a los números generados.
    f) Se actualiza el estado del autómata.
    g) Se repite el proceso desde el literal *b*) hasta tener una sucesión del tamaño del mensaje a encriptar.
2. Se escribe el mensaje a cifrar como una sucesión de *bits* utilizando la representación ASCII de los caracteres.
3. Se aplica la función *xor bit* a *bit* entre el mensaje a encriptar y la sucesión cifrante. La salida de este procedimiento es el mensaje encriptado.

El funcionamiento del método 2 y el método 3 es muy similar, por eso no tendría sentido aplicarle de nuevo las cinco pruebas estadśiticas. Lo que se cambió en el método 3 fue la función generadora de números pseudo aleatorios y por tanto con la prueba chi cuadrado realizada anteriormente es suficiente.

Para el último método se aplicará la prueba estadística universal de Maurer, una prueba que por sus características permite identificar deficiencias en el diseño de los generadores. Las pruebas se harán con $L = 16$ y las claves, tanto del autómata principal como del nonce, serán generadas aleatoriamente. Las longitudes de las claves principales serán de 256, 512 y 1024 para cada una de las tres pruebas.

Para que un generador pase la prueba el valor del estadístico de Maurer debe estar en el intervalo (15.166878600164,15.167879399836) tomando como nivel de significancia $\alpha = 0{,}001$.

Nonce	Clave	Estadístico Maurer
27034	22181	15.1673424
27034	27221	15.16718953
26025	22181	15.16703695
26025	27221	15.16730973
27221	22181	15.16716196
27221	27221	15.16707218

Cuadro 4: Resultados Prueba de Maurer con clave de 256 bits

Nonce	Clave	Estadístico Maurer
27034	22181	15.16739664
27034	27221	15.16753921
26025	22181	15.16708904
26025	27221	15.16726614
27221	22181	15.16714138
27221	27221	15.16717214

Cuadro 5: Resultados Prueba de Maurer con clave de 512 bits

Nonce	Clave	Estadístico Maurer
27034	22181	15.16731348
27034	27221	15.1673144
26025	22181	15.16764209
26025	27221	15.16747683
27221	22181	15.16732481
27221	27221	15.16720296

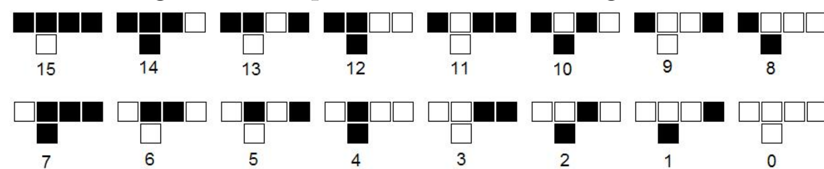
Cuadro 6: Resultados Prueba de Maurer con clave de 1024 bits

3.2. CRIPTOANÁLISIS

3.2.1. Claves Débiles

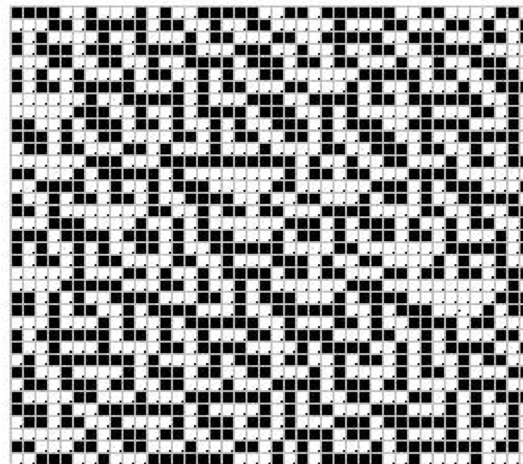
Durante el desarrollo del proyecto se vieron algunos comportamientos extraños que sucedían al tratar de evolucionar autómatas con un regla específica. Así éste hubiera pasado las pruebas con los margenes definidos, al escribir como semilla una repetición de los mismos caracteres, la evolución del autómata se volvía periódica, o en los casos más extremos estacionaria.

Figura 23: Representación de la Regla 21910



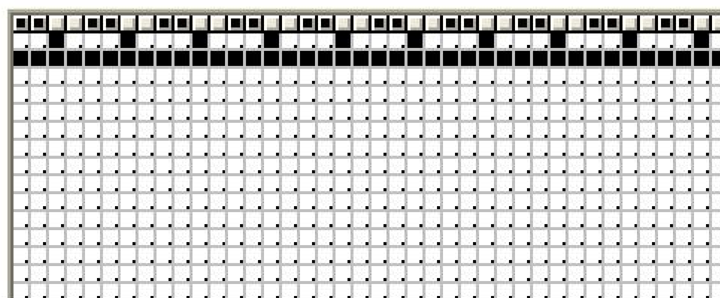
En la figura 23 se muestra la representación de la regla 21910, una de las muchas que pasaron las primeras pruebas estadísticas y por tanto pueden considerarse como criptograficamente seguras.

Figura 24: Evolución de la regla 21910 con una semilla aleatoria



En la figura 24 se presenta la evolución de la misma regla cuando se usa una semilla aleatoria. Se puede observar que no genera ningun tipo de patrones periódicos ni estacionarios. Por otro lado si la semilla tiene ciertas condiciones específicas (una repetición

Figura 25: Evolución de una clave débil en la regla 21910



de vecindades que siempre producen el mismo estado) el autómata pierde todas sus características de aleatoriedad. Éste comportamiento se muestra en la figura 25.

A pesar de las graves consecuencias que se presentarían al utilizar una clave débil para encriptar un mensaje, existen soluciones muy sencillas y efectivas que se pueden implementar. Por ejemplo se puede validar la clave cuando es ingresada, de tal manera que no se puedan usar las que tengan más de tres caracteres iguales consecutivos, o aquellas con patrones repetitivos que puedan dar lugar a una clave débil. Otra solución menos sencilla implicaría tener un sistema de generación aleatoria de claves, sin embargo se necesitaría también un sistema de distribución de las mismas, para que el usuario no tuviera que memorizar una sarta de caracteres extraños cada vez que necesitara encriptar algo.

3.2.2. Fuerza Bruta

La seguridad en un criptosistema simétrico está en función de dos cosas: la fortaleza del algoritmo generador y la longitud de la clave. No importa que tan fuerte sea el algoritmo, si la clave es débil[7], el criptosistema tambien lo será.

Para poner un ejemplo, supongamos que la fortaleza del algoritmo es perfecta. Esto es extremadamente difícil de lograr en la práctica pero por razones del ejemplo se justifica. Perfecto quiere decir no hay mejor manera de romper el criptosistema sino tratar cada posible clave con un ataque de fuerza bruta.

Para comenzar este ataque, un criptoanalista necesita una pequeña cantidad de texto cifrado y el correspondiente texto claro; un ataque por fuerza bruta es del tipo de texto claro conocido. Hay muchas maneras por las cuales un criptoanalista puede conseguir estos recursos; por ejemplo, el archivo encriptado puede ser un documento de Word, o un archivo de directorio de Unix, una imagen TIFF. Todos estos formatos

[7]Se refiere a claves muy cortas o con patrones repetitivos.

tienen cabeceras predefinidas y son iguales en todos los archivos.

Calcular la complejidad de un ataque por fuerza bruta es sencillo. Si la clave tiene 8 *bits* de longitud hay 2^8 o 256 posibles claves. Entonces tomará 256 intentos encontrar la clave correcta, con una probabilidad del 50 % de encontrarla en la mitad de los intentos. Si la clave tiene 56 *bits* entonces hay 2^{56} posibles claves. Suponiendo que una supercomputadora puede probar un millón de claves por segundo, le tomaría 2285 años encontrar la clave correcta. Si la clave tiene 64 *bits* le tomará, a la misma supercomputadora, 585.000 años encontrar la clave correcta entre las 2^{64} posibilidades. Si la clave tiene 128 *bits* tomará 10^{25} años. Teniendo en cuenta que el universo tiene 10^{10} años, 10^{25} es bastante tiempo.

La propuesta inicial en el proyecto era que la longitud de la clave fuera variable, sin embargo esto permitía que se usaran claves muy cortas y por consiguiente inseguras. Para solucionar esta situación y evitar que el usuario tenga que escribir y memorizar claves muy largas se propone una expansión de la clave de tal manera que si se trata de usar una clave corta, antes de empezar a encriptar, la misma se alargue de tal manera que llegue hasta los 256 *bits*, longitud que se cree suficiente para que un ataque por fuerza bruta sea inviable.

3.2.3. Inversión de un Autómata Celular

Meier y Staffelbach proponen en [10] que utilizando un sistema basado en autómatas celulares, al conocer N valores de la sucesión cifrante (por medio de un ataque de texto claro) se puede conocer la clave (de tamaño N) en su totalidad.

En la misma publicación se dice que basta conocer los valores de dos celdas adyacentes del autómata para reconstruir toda la evolucion del automata. El criptoanálisis es un ataque de fuerza bruta no a la semilla, sino a la sucesión cifrante, generando aleatoriamente las celdas que deberían estar a la derecha o a la izquierda. Esta forma de ataque es más eficiente y reduce considerablemente el tiempo que toma el ataque de fuerza bruta clásico, pues hay sucesiones que tienen mayor probabilidad de ser que otras.

El método 1 es débil a este ataque, aunque por la vecindad es menos efectivo pues para generar todos los valores es necesario conocer las 2 celdas adyacentes. El método 2 y el método 3 ya no presentan esta debilidad, el ataque solo es efectivo si la sucesión cifrante es tomada de la misma columna, al ser aleatoria la celda que se escoge para la sucesión cifrante, este ataque es totalmente inefectivo.

4. DESARROLLO DE LA APLICACIÓN

Para la implementación de la aplicación, se crearon 3 tipos de datos, "binnum" que representa un arreglo de tamaño dinámico (el tamaño se declara en tiempo de ejecución) de tipo *byte*[8], "rannum" que es un arreglo de tamaño 32 del tipo *double*[9] y "cuatrobytes" un arreglo de tamaño 4 del tipo *byte*, el primero se usará para representar las sucesiones binarias, el segundo y el tercero para la implementación del método 3.

Delphi trae implementada por defecto la función *xor*, esta se hace a nivel de *bits* en dos variables del mismo tipo. Al ser el *byte* la unidad básica de almacenamiento de información, se utilizará para el proceso de cifrado, es decir, se leerá un *byte* del archivo a cifrar, se generará, utilizando los autómatas celulares, un *byte* de sucesión cifrante y a estos dos se les aplicará la función *xor*.

```
binnum=array of byte;
rannum=array[0..31] of double;
cuatrobytes=array[0..4] of byte;
```

4.1. LA CLASE AUTÓMATA

En la aplicación no es necesario tener presentes todos los estados que muestra el autómata celular a lo largo de sus evoluciones, con conocer el último y el primero es suficiente. Por este motivo, la clase incluye dos propiedades de tipo "binnum"; semilla y evolución. La clase también incluye otra propiedad de tipo "binnum" en la que se guardan las reglas que rigen al autómata, donde el índice es la representación decimal de la regla y el contenido es el estado al que pasa la célula (ver Fig. ALGUNA FIGURA). Para algunos cálculos es necesario tener presente el tamaño de la semilla por lo que este también es guardado como la propiedad "largo". Por cuestiones de eficiencia también se hizo necesario incluir una propiedad en la que se guarda el estado al que va a pasar el autómata en cada evolución "VEvolucion". Esta clase implementa además del constructor, 4 métodos, los cuales son usados para la implementación de cada uno de los métodos de encriptación descritos en la propuesta.

```
  TAutomataCelular = class(TObject)
  public
    largo:integer;
    regla:binnum;
```

[8]números enteros sin signo de 8 *bits*
[9]números reales de 64 *bits*

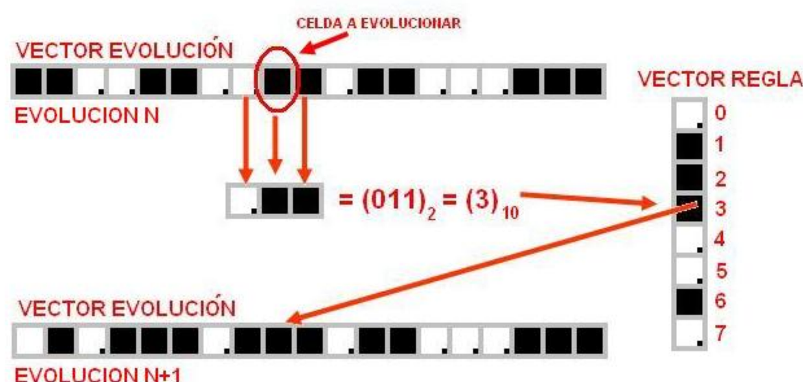
```
    semilla:binnum;
    Evolucion:binnum;
    VEvolucion:binnum;
    procedure evolucionar();
    constructor create(Vregla:binnum;Vsemilla:binnum);
    function DevolverUnByte(numerobits:byte;Nonce:TAutomataCelular):byte;
    function DevolverUnByteSinNonce():byte;
    function Devolver32Aleatorios(numerobits:byte):rannum;
    function DevolverCuatroBytes(numerobits:byte;
      nonce:TAutomatacelular):cuatrobytes;
  end;
```

El método constructor del objeto (*create*) recibe como parámetros dos vectores, la semilla y la regla, el primero se asigna a los campos evolución y semilla, y el segundo se asigna al campo regla. Además, en este método se evoluciona el autómata celular un número $\frac{(n-1)}{3}+1$ de veces siendo n la longitud de la semilla, de esta manera se asegura que el más mínimo cambio en la semilla (así sea solo un *bit*) afecte todo el proceso de evolución del autómata (Fig. 20).

```
  constructor create(Vregla:binnum;Vsemilla:binnum);
  var
    i:integer;
  begin
    largo:=length(vsemilla);
    setlength(semilla,largo);
    setlength(regla,16);
    setlength(Evolucion,largo);
    setlength(VEvolucion,largo);
    semilla:=Vsemilla;
    Evolucion:=Vsemilla;
    regla:=Vregla;
    for i:= 0 to trunc((length(semilla)-1)/3 +1) do
      evolucionar();
  end;
```

El método evolucionar, como su nombre lo dice, actualiza el estado del vector evolución siguiendo las reglas establecidas. Como todo el cambio de estado del autómata celular es simultáneo, este método guarda primero el cambio de cada celda del autómata en un vector temporal “vevolucion”, para que al final del método, se asigne al vector “evolucion”. Para evolucionar cada celda “evolucion[i]”, se guarda el estado de ésta y de su vecindad en el vector “temp”, este vector lo interpretamos como un número binario (ver Fig. 26), que en representación decimal es la posición de el vector “regla” donde

Figura 26: Funcionamiento del procedimiento evolucionar



está el estado al cual debe pasar la celda, que a su vez es guardado en "vevolucion[i]". Después de este proceso se asigna a "evolucion" el vector "Vevolucion".

```
procedure evolucionar();
var
  j,i:integer;
  temp:array [0..3] of byte;
begin
  temp[0]:=Evolucion[largo-1];
  temp[1]:=Evolucion[largo-2];
  temp[2]:=Evolucion[0];
  temp[3]:=Evolucion[1];
  vevolucion[0]:=regla[8*temp[0]+4*temp[1]+2*temp[2]+temp[3]];
  temp[0]:=Evolucion[largo-1];
  for i:=1 to 3 do
    temp[i]:=Evolucion[i-1];
  vevolucion[1]:=regla[8*temp[0]+4*temp[1]+2*temp[2]+temp[3]];
  temp[0]:=Evolucion[largo-3];
  temp[1]:=Evolucion[largo-2];
  temp[2]:=Evolucion[largo-1];
  temp[3]:=Evolucion[0];
  vevolucion[largo-1]:=regla[8*temp[0]+4*temp[1]+2*temp[2]+temp[3]];
  for i:=2 to largo-2 do
  begin
    temp[0]:=Evolucion[i-2];
    temp[1]:=Evolucion[i-1];
    temp[2]:=Evolucion[i];
    temp[3]:=Evolucion[i+1];
    vevolucion[i]:=regla[8*temp[0]+4*temp[1]+2*temp[2]+temp[3]];
```

```
    end;
    Evolucion:=vevolucion;
  end;
```

El procedimiento "DevolverUnByteSinNonce" es la base para la implementación del método 1 o cifrado clásico, éste devuelve un número entre 0 y 255 que es la representación decimal de un número binario de 8 dígitos. Para entender la forma en que este método trabaja recordemos que un número binario $a_0, a_1, a_2, a_3, a_4, ..., a_n$ se expresa en notación decimal de la siguiente forma $a_0 * 2^0 + a_1 * 2^1 + a_2 * 2^2 + a_3 * 2^4 + ... + a_n * 2^n$. En el bucle principal se evoluciona el autómata, se obtiene el contenido de la celda 20 y con este se forma el número.

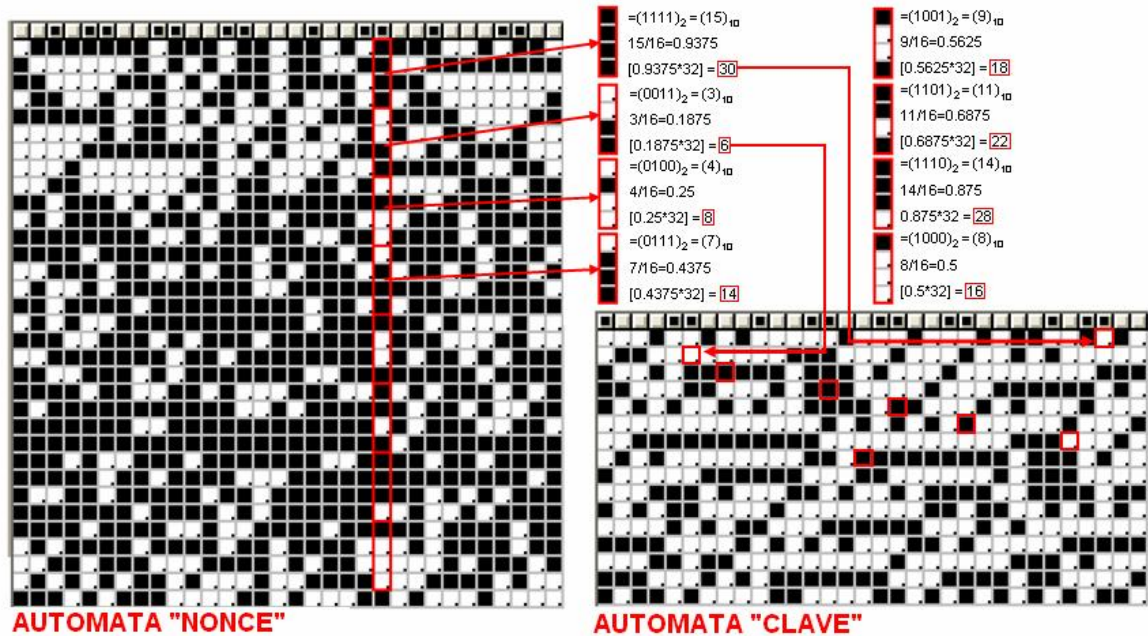
```
function DevolverUnByteSinNonce():byte;
  var
    j:integer;
    valor,potencia:byte;
  begin
    valor:=0;
    potencia:=1;
    for j:=0 to 7 do
    begin
      evolucionar();
      valor:=Evolucion[20]*potencia+valor;
      potencia:=potencia*2;
    end;
    devolverunbytesinnonce:=valor;
  end;
```

El procedimiento "DevolverUnByte" es la base de la implementación del método 2, este recibe como parámetro el número de *bits* con el que va a trabajar el generador pseudo aleatorio y el autómata que se va a usar como *nonce*. Este método funciona básicamente igual a "DevolverUnByteSinNonce", pero en vez de elegir siempre la celda 20 se utiliza el generador pseudo aleatorio para elegir la celda. Este generador pseudo aleatorio retorna un número entre 0 y $2^numerobits - 1$ con base en la celda 20 del autómata *nonce* el cual se divide en $2^numerobits$ para obtener un número en el intervalo $[0, 1)$.

El número devuelto por el generador se multiplica por el largo de la evolución para obtener la celda a utilizar.27

```
  function DevolverUnByte(numerobits:byte;Nonce:TAutomataCelular):byte;
  var
    i,j,num:integer;
```

Figura 27: Función Devolverunbyte




```
  valor,potencia,bin:byte;
  numero,potencia2:int64;
  r:double;
begin
  valor:=0;
  potencia:=1;
  for j:=0 to 7 do
  begin
    potencia2:=1;
    numero:=0;
    for i:=0 to numerobits do
    begin
      Nonce.evolucionar();
      numero:=potencia2*nonce.evolucion[20]+numero;
      potencia2:=potencia2*2;
    end;
    evolucionar();
    num:=trunc((largo)*(numero/(potencia2)));
```

```
    bin:=Evolucion[num];
    valor:=bin*potencia+valor;
    potencia:=potencia*2;
  end;
  devolverunbyte:=valor;
end;
```

"Devolver32aleatorios" funciona de la misma forma que el generador pseudo aleatorio explicado anteriormente, simplemente que este método devuelve 32 números pseudo aleatorios. La ventaja de este método es que requiere *31*numerobits* veces menos evoluciones.

```
function Devolver32Aleatorios(numerobits:byte):rannum;
var
  i,j,k:integer;
  valor,potencia,bin:byte;
  numero,potencia2:int64;
  res:rannum;
begin
  potencia2:=1;
  for i:=0 to 31 do
    res[i]:=0;
  for i:=0 to numerobits do
  begin
    j:=0;
    k:=0;
    evolucionar();
    while( J<256) do
    begin
      res[k]:=potencia2*evolucion[j]+res[k];
      inc(k,1);
      inc(j,8);
    end;
    potencia2:=potencia2*2;
  end;
  for i:=0 to 31 do
  begin
    res[i]:=res[i]/(potencia2);
  end;
  result:=res;
end;
```

"DevolverCuatroBytes" funciona igual que "DevolverUnByte" simplemente que utiliza la función "Devolver32Aleatorios" como generador pseudo aleatorio para obtener

las 32 posiciones que forman 4 *bytes*. De una sola vez.

```
  Function DevolverCuatroBytes(numerobits:byte;
    nonce:TAutomatacelular):cuatrobytes;
  var
    i,j,k:integer;
    num:cuatrobytes;
    potencia:int64;
    aleatorios:rannum;
  begin
    potencia:=1;
    aleatorios:=nonce.Devolver32Aleatorios(numerobits);
    for i:=0 to 3 do
      num[i]:=0;
    k:=0;
    for j:=0 to 3 do
    begin
      evolucionar();
      potencia:=1;
      for i:=0 to 7 do
      begin
        num[j]:=num[j]+potencia*(evolucion[trunc((largo-1)*(aleatorios[k]))]);
        potencia:=potencia*2;
        inc(k,1);
      end;

    end;
    result:=num;
  end;
end.
```

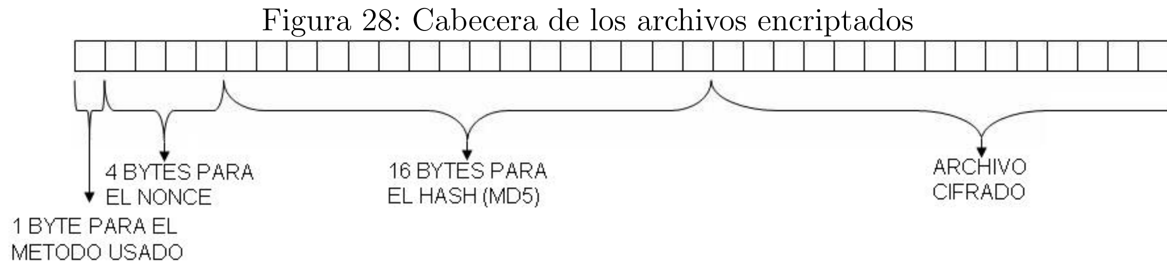
4.2. IMPLEMENTACIÓN DE LOS MÉTODOS

Cada método fue implementado en un hilo, el constructor de cada uno de estos salvo algunos pequeños cambios (en el método 1 no es necesario inicializar el nonce) es el mismo. Éste recibe como parámetros 5 variables, el formulario que lo llama, “Owner”, la barra de progreso que debe usar el hilo, “ProgressB”, la clave secreta para el cifrado, “clave”, la ubicación del archivo, “ubicacion”, y una variable que indica si vamos a descifrar o a cifrar, “tipo”, que toma los valores “e”(cifrar) y “d” (descifrar). Si la variable “tipo” es “e”, entonces el constructor inicializa la función *randomize* de Delphi, abre los archivos de entrada y salida, y, escribe la cabecera del archivo de salida. Si es

“d” entonces Delphi lee la cabecera del archivo de entrada y la asigna a las variables correspondientes. Después de este proceso se inicializan los autómatas a utilizar Y las variables necesarias para el funcionamiento del proceso.

La cabecera del archivo es la siguiente:

Figura 28: Cabecera de los archivos encriptados



Esta se incluye manera que el programa tenga la información necesaria para poder descifrarlo, que es: el método usado, el nonce y el codigo hash del archivo. El método se usa para saber de que forma lo vamos a descifrar, si utilizando el cifrado clasico (1), el cifrado con nonce (2) o el cifrado con nonce mejorado (3). El nonce se envía como texto claro y el hash del archivo original se envia para comprobar que el mensaje no haya sido modificado.

El ciclo principal del proceso de cifrado es simple, hasta que se llegue al final de el archivo, lea un byte, obtenga un byte usando autómatas celulares, realice la operación Xor entre estos dos y escriba el byte obtenido en el archivo de salida.

- Ciclo principal del Método 1

```
repeat
      blockread(archi,t,1);
      t:=t xor automataclave.DevolverUnByteSinNonce();
      blockwrite(archo,t,1);
      progressbar.StepIt;
      inc(numero,1);
until (terminated=true)or(numero=filesize(archi));
```

- Ciclo principal Método 2

```
h:=15;
    repeat
      blockread(archi,t,1);
```

```
    t:=t xor    AutomataClave.DevolverUnByte(h, AutomataNonce);
    blockwrite(archo,t,1);
    progressbar.StepIt;
    inc(numero,1);
  until (terminated=true)or(numero=filesize(archi));
```

- Ciclo principal Método 3

```
repeat
    k:=0;
    bytes:=automataclave.DevolverCuatroBytes(h, automatanonce);
    repeat
      blockread(archi,t,1);
      t:=t xor bytes[k];
      blockwrite(archo,t,1);
      inc(k,1);
      progressbar.StepIt;
      inc(numero,1);
    until (k=4) or(numero=filesize(archi));
  until (terminated=true)or(numero=filesize(archi));
```

4.3. ENTORNO VISUAL

Ventana principal (Fig. 29): La ventana principal nos ofrece un menú con tres opciones, cifrar texto, cifrar archivos y descifrar. Además nos ofrece acceso a la ayuda y al "acerca de".

Figura 29: Diálogo Principal

Cifrar Archivo (Fig. 30): Esta opción cifra un archivo indiferentemente del tipo, creando un nuevo archivo en la misma ubicación y de la misma extensión que el original

pero con una “e” al final. Al seleccionar esta opción, aparece un nuevo formulario solicitando los datos para el cifrado, esto son: el archivo a cifrar, la contraseña y el método de cifrado.

Una vez están llenos todos los campos (Fig 33), se procede a hacer clic en el botón “Cifrar”, ante lo cual empezara el proceso. El progreso de este se podrá observar en la barra que esta debajo de el botón cifrar. Además, aparece en el formulario un botón “Cancelar” de manera que en cualquier momento se pueda detener el cifrado. Otra forma de cancelar el proceso es cerrando el formulario.

Al terminar el proceso de cifrado, aparece una ventana con un mensaje de texto informándolo y el formulario se cierra automáticamente. (Fig. 36)

Cifrar texto: Esta opción cifra un texto dado a un archivo de extensión .txte Al hacer clic en esta opción, aparece en la ventana principal un espacio para escribir el texto a cifrar, junto con un botón “Cifrar Texto” para continuar con el proceso. (Fig. 37)

Una vez escrito el texto a cifrar, se hace *click* en el botón “cifrar texto”, y aparece un nuevo formulario en el que se solicita la información para el cifrado (contraseña, archivo de destino, tipo de cifrado). El cual funciona igual al formulario para cifrar archivos con la excepción de la extensión del archivo de destino es .txte y que si este no existe se tiene la opción de crearlo.(Fig. 38)

Descifrar: Esta opción obtiene el archivo original de un archivo cifrado. Al seleccionar esta opción, aparece un nuevo formulario solicitando el nombre de archivo y contraseña. (Fig. 39)

Este proceso después de descifrar verifica que el hash del archivo cifrado y el descifrado sea el mismo, si no lo es, borra el archivo descifrado y advierte al usuario de la situación.

Figura 30: Cifrar Archivo

Figura 31: Diálogo Encriptar Archivo

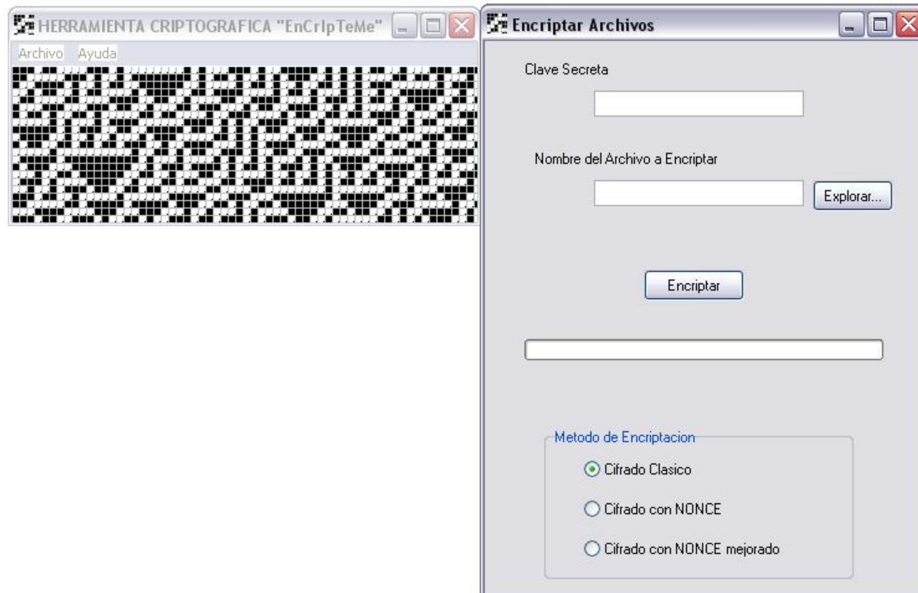


Figura 32: Selección del archivo a encriptar

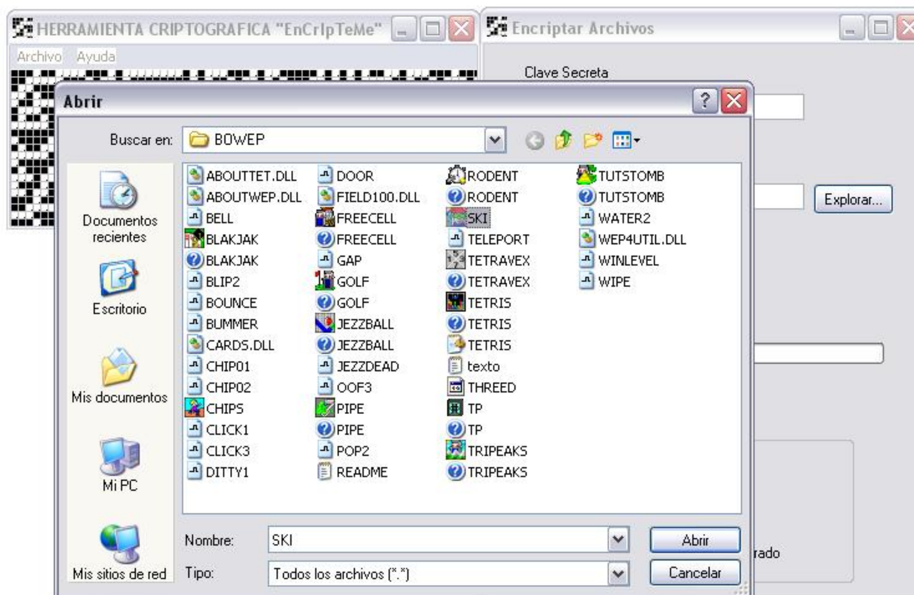


Figura 33: Archivo Seleccionado para Encriptar

Figura 34: Ejecución del Encriptado

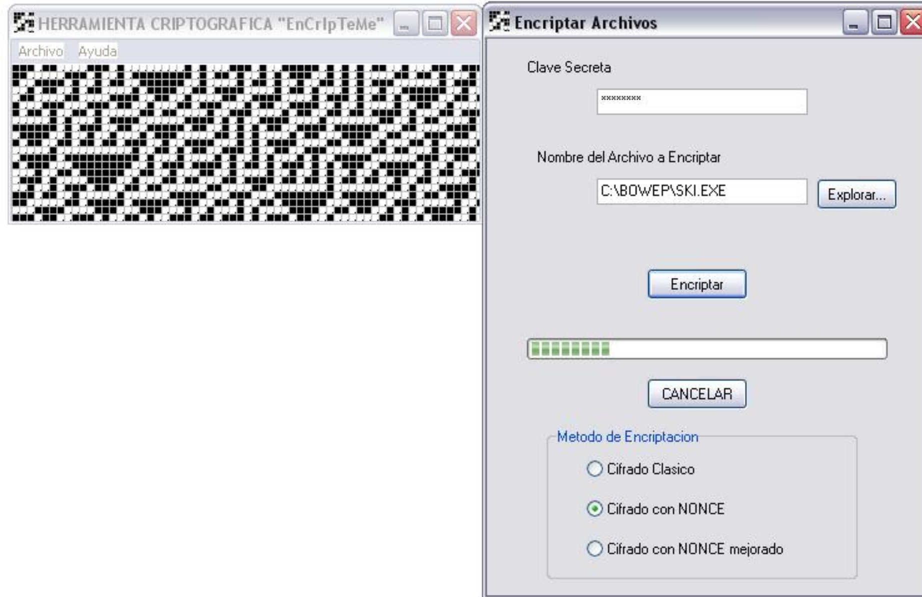


Figura 35: Cancelación del Proceso

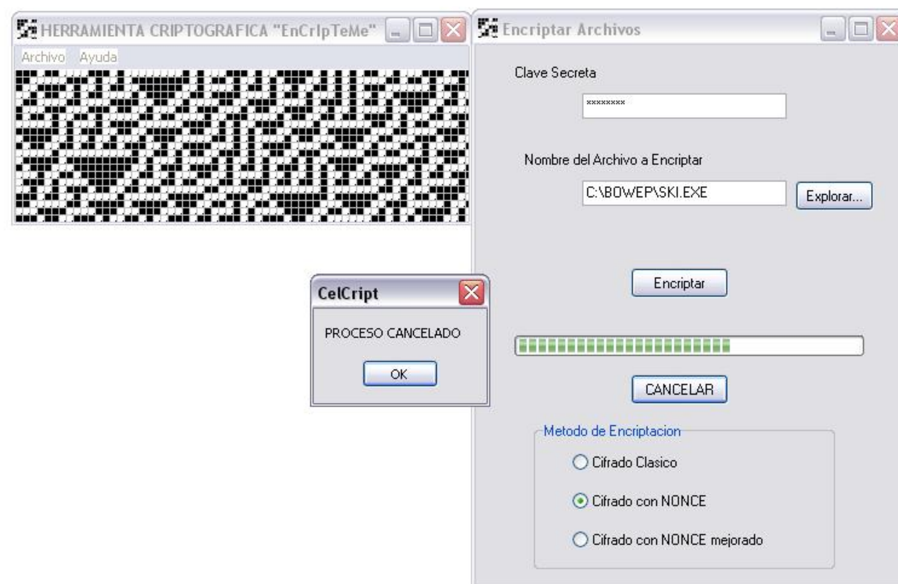


Figura 36: Proceso Terminado

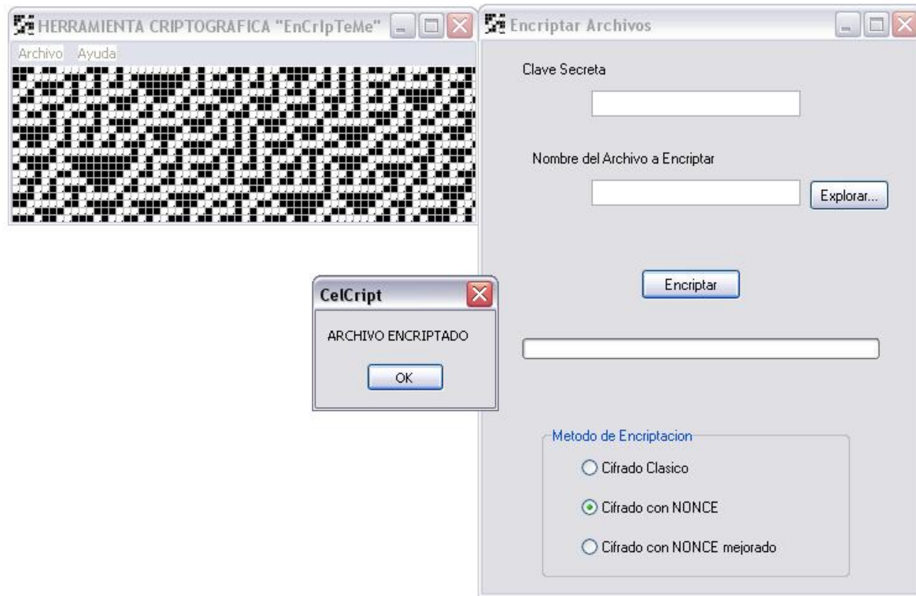


Figura 37: Diálogo para escribir el texto a encriptar

Figura 38: Diálogo Encriptar Texto

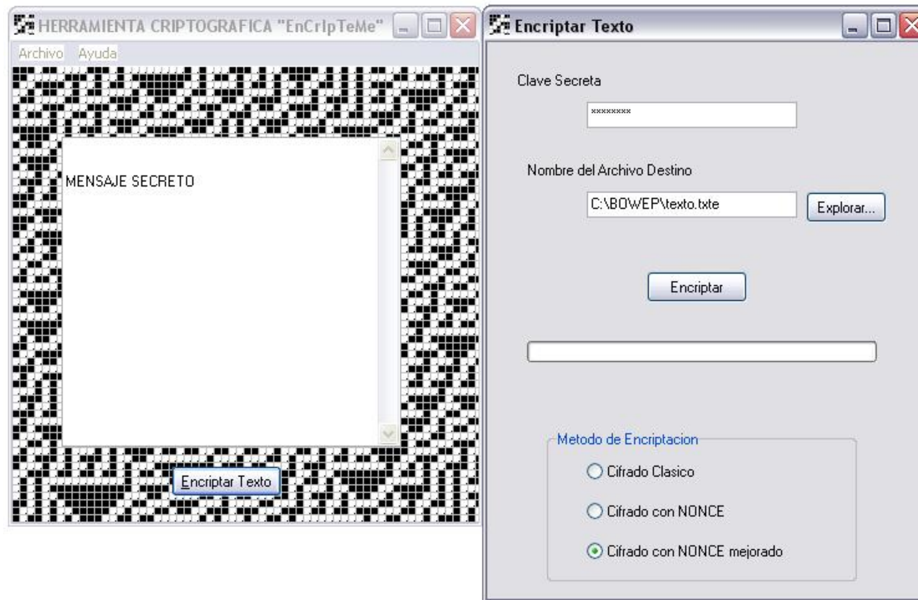


Figura 39: Diálogo Desencriptar

5. CONCLUSIONES

1. Utilizar dos claves, una del usuario y otra aleatoria, permitió que la primera pudiera ser usada varias veces y con esto evitar la molestia de tener que estar cambiando de claves cada vez que se fuera a encriptar un mensaje. Esto era necesario debido a la alta vulnerabilidad de un sistema de clave secreta al ataque de clave reusada.

2. La utilización del nonce no debería incidir en la velocidad de encriptación y desencriptación, se hizo necesario un replanteamiento del algoritmo para mejorar su velocidad con muy buenos resultados.

3. Los algoritmos simétricos por si solos no permiten la validación en la integridad de los mensajes enviados, por eso se usó la función hash MD5, que permite crear una "huella digital" del mensaje antes de encriptarlo y después de desencriptarlo. Ahora es posible decir que el mensaje recibido fué, efectivamente el enviado y que no sufrió modificaciones durante la transmisión.

4. Se desarrolló una función generadora de números pseudo aleatorios basada en Autómatas Celulares que permitió la elección de celdas de diferentes columnas lo que aumentó el porcentaje de sucesiones de *bits* que pasaban las pruebas estadísticas.

5. La utilización de dos claves, un generador de números pseudo aleatorios y autómatas de vecindad 4 permitío aumentar la seguridad y disminuir las debilidades que presentaba el sistema de cifrado en flujo basado en autómatas celulares propuesto por Wolfram.

6. RECOMENDACIONES

1. Se debe implementar un generador aleatorio de claves para evitar que el usuario digite claves débiles y ponga en peligro su propia información.

2. La aplicación criptográfica puede complementarse con un sistema seguro de distribución de claves que aumentaría significativamente el atractivo comercial pues podría convertirse en un medio seguro para comunicaciones confidenciales.

3. A lo largo del desarrollo del proyecto se encontró en diferentes fuentes que la implementación de los autómatas celulares en hardware es posible y relativamente sencilla. Lo anterior aumentaría significativamente la velocidad de encriptación/desencriptación manteniendo la seguridad.

4. Los autómatas celulares son herramientas computacionales que hacen parte de la inteligencia artificial basada en modelos biológicos, como tal, se pueden encontrar aplicaciones en muchos campos diferentes a la criptografía, por eso se recomienda ampliar el estudio de estas herramientas para poder encontrar otras alternativas de solución a diferentes tipos de problemas.

ANEXO A

# AC	P_1	P_2	P_3	P_4	P_5
10710	98	97	98	96	100
22870	97	97	96	96	96
26021	96	99	98	97	97
38246	97	98	97	98	96

Cuadro 7: Autómatas que pasaron la prueba con el generador 11730

# AC	P_1	P_2	P_3	P_4	P_5
22934	96	96	99	96	96
26982	96	96	96	96	96
27225	97	96	98	97	97
38246	97	97	96	97	98
39333	97	99	97	96	99

Cuadro 8: Autómatas que pasaron la prueba con el generador 15420

# AC	P_1	P_2	P_3	P_4	P_5
18360	96	99	96	98	97
38501	96	96	96	97	99
38553	96	99	97	98	96

Cuadro 9: Autómatas que pasaron la prueba con el generador 21846

# AC	P_1	P_2	P_3	P_4	P_5
18360	97	96	98	98	98
26217	96	96	97	96	96
34695	96	96	96	96	97

Cuadro 10: Autómatas que pasaron la prueba con el generador 21913

# AC	P_1	P_2	P_3	P_4	P_5
4590	96	97	96	97	96
23209	96	97	98	97	97

# AC	P_1	P_2	P_3	P_4	P_5
43370	98	97	97	98	96

Cuadro 11: Autómatas que pasaron la prueba con el generador 21925

# AC	P_1	P_2	P_3	P_4	P_5
21865	98	96	96	96	96
27049	96	97	96	97	99
42394	98	96	98	96	96

Cuadro 12: Autómatas que pasaron la prueba con el generador 22105

# AC	P_1	P_2	P_3	P_4	P_5
22870	98	97	96	96	96
34695	99	99	96	97	98
39514	96	96	97	96	97

Cuadro 13: Autómatas que pasaron la prueba con el generador 22118

# AC	P_1	P_2	P_3	P_4	P_5
37485	98	97	97	96	96
42346	97	96	97	97	96
43669	96	98	96	96	96
46665	97	97	96	97	98

Cuadro 14: Autómatas que pasaron la prueba con el generador 22122

# AC	P_1	P_2	P_3	P_4	P_5
28050	99	97	96	96	96
38245	97	96	99	98	98
43354	99	97	97	96	96

Cuadro 15: Autómatas que pasaron la prueba con el generador 22166

# AC	P_1	P_2	P_3	P_4	P_5
22869	97	98	96	98	97

# AC	P_1	P_2	P_3	P_4	P_5
22937	97	96	99	96	96
26217	97	97	98	97	98
42645	98	97	99	96	97
43609	98	97	97	98	98
44370	98	99	96	98	98

Cuadro 16: Autómatas que pasaron la prueba con el generador 22185

# AC	P_1	P_2	P_3	P_4	P_5
23141	96	96	96	96	98
23145	97	96	96	96	96
26261	98	99	96	99	98
34680	98	98	97	96	97
38553	96	98	96	98	97
57630	97	96	96	96	97

Cuadro 17: Autómatas que pasaron la prueba con el generador 22937

# AC	P_1	P_2	P_3	P_4	P_5
22870	97	98	98	96	97
39334	97	98	98	96	97
42390	96	97	96	97	96

Cuadro 18: Autómatas que pasaron la prueba con el generador 22953

# AC	P_1	P_2	P_3	P_4	P_5
38246	99	98	97	96	96
39514	97	98	98	96	99
42329	96	99	96	99	97

Cuadro 19: Autómatas que pasaron la prueba con el generador 23126

# AC	P_1	P_2	P_3	P_4	P_5
38246	98	96	96	96	97
38501	98	96	98	96	96
43366	98	97	96	96	97

# AC	P_1	P_2	P_3	P_4	P_5
46665	96	99	96	96	96

Cuadro 20: Autómatas que pasaron la prueba con el generador 23129

# AC	P_1	P_2	P_3	P_4	P_5
22953	97	99	97	96	98
23125	97	98	100	96	96
25961	98	99	96	96	96
26201	96	97	96	96	97
37485	98	96	97	97	96

Cuadro 21: Autómatas que pasaron la prueba con el generador 23141

# AC	P_1	P_2	P_3	P_4	P_5
9690	98	96	97	98	96
22954	96	96	98	96	97
23146	96	97	96	97	98
42586	96	96	96	98	96

Cuadro 22: Autómatas que pasaron la prueba con el generador 23142

# AC	P_1	P_2	P_3	P_4	P_5
25942	96	96	96	98	97
26025	99	97	96	99	96
39254	96	97	96	98	97

Cuadro 23: Autómatas que pasaron la prueba con el generador 23146

# AC	P_1	P_2	P_3	P_4	P_5
22869	98	97	96	96	99
26261	96	96	96	96	96
27034	100	98	96	96	96

Cuadro 24: Autómatas que pasaron la prueba con el generador 25941

# AC	P_1	P_2	P_3	P_4	P_5
25957	96	97	96	98	98
38246	97	98	97	96	100
42390	96	98	97	96	96
43418	98	97	97	97	96

Cuadro 25: Autómatas que pasaron la prueba con el generador 25962

# AC	P_1	P_2	P_3	P_4	P_5
25962	97	97	96	96	97
26198	96	96	97	97	97
26969	97	98	96	98	99
43609	97	96	96	96	97

Cuadro 26: Autómatas que pasaron la prueba con el generador 26198

# AC	P_1	P_2	P_3	P_4	P_5
21862	96	97	98	98	97
27046	99	98	97	96	97
38553	96	97	99	96	96
39254	96	97	99	96	98

Cuadro 27: Autómatas que pasaron la prueba con el generador 26966

# AC	P_1	P_2	P_3	P_4	P_5
21865	96	97	98	96	96
22121	96	96	98	96	98
26198	98	97	96	97	96
26970	97	97	97	97	97
39269	96	97	96	97	97
42393	96	96	96	98	97

Cuadro 28: Autómatas que pasaron la prueba con el generador 27221

# AC	P_1	P_2	P_3	P_4	P_5
22121	96	96	96	97	96
26985	96	99	96	97	97

# AC	P_1	P_2	P_3	P_4	P_5
44370	97	98	98	97	96

Cuadro 29: Autómatas que pasaron la prueba con el generador 27225

# AC	P_1	P_2	P_3	P_4	P_5
22953	96	96	98	96	96
25962	98	97	98	96	97
38310	97	96	96	96	98
39274	96	97	97	97	98

Cuadro 30: Autómatas que pasaron la prueba con el generador 27226

# AC	P_1	P_2	P_3	P_4	P_5
17595	98	98	97	97	99
21926	98	97	96	98	97
22938	97	99	97	97	96
39333	97	98	96	98	96

Cuadro 31: Autómatas que pasaron la prueba con el generador 27242

# AC	P_1	P_2	P_3	P_4	P_5
39269	98	99	96	97	97
42597	96	97	97	97	98
57630	96	97	96	96	97

Cuadro 32: Autómatas que pasaron la prueba con el generador 27290

# AC	P_1	P_2	P_3	P_4	P_5
11730	98	96	96	97	96
22122	96	97	97	97	96
22869	96	97	97	98	97
26202	97	97	98	98	97
43625	99	97	98	98	96

Cuadro 33: Autómatas que pasaron la prueba con el generador 27305

# AC	P_1	P_2	P_3	P_4	P_5
23141	96	97	100	97	98
26262	96	96	96	96	96
42645	96	97	97	97	96

Cuadro 34: Autómatas que pasaron la prueba con el generador 34425

# AC	P_1	P_2	P_3	P_4	P_5
17595	96	97	97	98	97
21865	98	98	96	99	96
43609	97	96	96	98	97

Cuadro 35: Autómatas que pasaron la prueba con el generador 38245

# AC	P_1	P_2	P_3	P_4	P_5
23145	96	98	96	96	97
26969	96	96	96	96	97
34680	97	98	98	96	97
39258	96	97	99	99	98

Cuadro 36: Autómatas que pasaron la prueba con el generador 38249

# AC	P_1	P_2	P_3	P_4	P_5
23146	96	96	97	96	97
26265	98	98	96	96	98
27289	96	96	96	97	96
38501	98	98	97	96	99

Cuadro 37: Autómatas que pasaron la prueba con el generador 38293

# AC	P_1	P_2	P_3	P_4	P_5
4590	97	98	96	97	98
26985	98	98	96	98	96
42649	96	97	98	96	96

Cuadro 38: Autómatas que pasaron la prueba con el generador 38297

# AC	P_1	P_2	P_3	P_4	P_5
22185	96	96	96	100	98
26213	98	97	97	96	98
38501	96	97	97	98	99
39333	96	97	97	96	96

Cuadro 39: Autómatas que pasaron la prueba con el generador 38310

# AC	P_1	P_2	P_3	P_4	P_5
27049	96	96	96	96	96
39526	97	96	98	99	97
43353	98	96	96	98	97

Cuadro 40: Autómatas que pasaron la prueba con el generador 38489

# AC	P_1	P_2	P_3	P_4	P_5
22121	99	98	96	97	97
22870	98	97	96	97	98
28050	97	97	98	96	97
39270	99	98	98	98	98

Cuadro 41: Autómatas que pasaron la prueba con el generador 38501

# AC	P_1	P_2	P_3	P_4	P_5
22185	96	96	96	97	97
26265	96	96	97	97	96
38501	96	98	97	97	97

Cuadro 42: Autómatas que pasaron la prueba con el generador 38506

# AC	P_1	P_2	P_3	P_4	P_5
21913	100	96	98	98	96
38249	98	97	98	96	97
38550	97	98	96	97	97
42597	97	96	96	96	96

Cuadro 43: Autómatas que pasaron la prueba con el generador 38550

# AC	P_1	P_2	P_3	P_4	P_5
22122	99	96	96	97	97
23142	97	98	98	98	97
27049	97	96	96	99	97

Cuadro 44: Autómatas que pasaron la prueba con el generador 38565

# AC	P_1	P_2	P_3	P_4	P_5
22185	97	96	99	96	96
26262	98	99	97	97	97
42601	96	98	96	96	96

Cuadro 45: Autómatas que pasaron la prueba con el generador 39253

# AC	P_1	P_2	P_3	P_4	P_5
26261	100	97	97	98	97
39573	96	97	96	97	96
43414	96	98	97	97	98
57630	96	97	96	96	96

Cuadro 46: Autómatas que pasaron la prueba con el generador 39257

# AC	P_1	P_2	P_3	P_4	P_5
21926	98	97	97	97	97
22870	97	99	96	96	96
22953	96	98	99	96	96
27046	96	98	99	97	98
39270	98	98	97	98	96

Cuadro 47: Autómatas que pasaron la prueba con el generador 39274

# AC	P_1	P_2	P_3	P_4	P_5
22870	96	98	97	96	98
38501	97	98	97	96	96
39269	98	96	96	96	96

Cuadro 48: Autómatas que pasaron la prueba con el generador 39317

# AC	P_1	P_2	P_3	P_4	P_5
9690	96	96	98	96	98
22938	100	98	96	96	96
22954	97	98	96	99	96
42585	96	97	96	96	96

Cuadro 49: Autómatas que pasaron la prueba con el generador 39510

# AC	P_1	P_2	P_3	P_4	P_5
21866	98	98	96	98	96
26985	97	98	97	98	96
43625	96	97	97	96	98

Cuadro 50: Autómatas que pasaron la prueba con el generador 39513

# AC	P_1	P_2	P_3	P_4	P_5
22181	97	99	96	97	97
38309	97	97	96	96	98
39573	97	98	97	96	99
46155	98	96	96	96	97

Cuadro 51: Autómatas que pasaron la prueba con el generador 39573

# AC	P_1	P_2	P_3	P_4	P_5
22121	99	99	96	97	97
27285	97	96	97	98	98
42394	97	96	97	96	96
43413	98	98	98	96	98

Cuadro 52: Autómatas que pasaron la prueba con el generador 39574

# AC	P_1	P_2	P_3	P_4	P_5
18360	99	99	96	96	97
21865	98	98	99	96	96
22105	97	97	96	98	96
34695	97	96	99	96	98

# AC	P_1	P_2	P_3	P_4	P_5

Cuadro 53: Autómatas que pasaron la prueba con el generador 39593

# AC	P_1	P_2	P_3	P_4	P_5
22954	98	98	97	97	97
38553	98	100	97	96	97
43669	97	97	98	97	96

Cuadro 54: Autómatas que pasaron la prueba con el generador 42326

# AC	P_1	P_2	P_3	P_4	P_5
22933	96	96	98	97	96
27289	99	99	96	99	96
38309	97	96	96	96	96
43366	98	96	96	96	96

Cuadro 55: Autómatas que pasaron la prueba con el generador 42329

# AC	P_1	P_2	P_3	P_4	P_5
22933	99	99	98	96	98
27046	96	96	97	96	96
39269	97	96	97	97	98
39333	97	98	99	97	96

Cuadro 56: Autómatas que pasaron la prueba con el generador 42393

# AC	P_1	P_2	P_3	P_4	P_5
6630	98	100	96	98	97
22186	98	98	96	98	98
22870	96	97	97	96	98
39258	96	97	96	97	98
39526	96	97	98	96	96
43349	96	96	97	96	96

Cuadro 57: Autómatas que pasaron la prueba con el generador 42394

# AC	P_1	P_2	P_3	P_4	P_5
23142	98	98	97	99	97
39254	96	96	96	96	98
39513	96	98	97	96	98

Cuadro 58: Autómatas que pasaron la prueba con el generador 42406

# AC	P_1	P_2	P_3	P_4	P_5
23141	96	97	97	96	96
39589	99	96	97	96	98
42346	98	96	96	96	96
43609	98	96	96	96	96

Cuadro 59: Autómatas que pasaron la prueba con el generador 42409

# AC	P_1	P_2	P_3	P_4	P_5
26217	96	96	99	96	96
26981	96	97	97	98	96
27285	96	97	97	96	98
43413	97	96	98	96	96
43669	98	97	97	97	96

Cuadro 60: Autómatas que pasaron la prueba con el generador 42585

# AC	P_1	P_2	P_3	P_4	P_5
23142	98	97	99	96	97
26277	97	96	98	97	96
39334	96	97	100	97	98

Cuadro 61: Autómatas que pasaron la prueba con el generador 42646

# AC	P_1	P_2	P_3	P_4	P_5
22122	99	98	96	97	97
22937	96	97	98	96	96
42597	98	96	97	96	97

Cuadro 62: Autómatas que pasaron la prueba con el generador 43353

# AC	P_1	P_2	P_3	P_4	P_5
26277	97	97	98	96	97
26981	99	98	97	96	99
27049	97	97	97	96	97
43353	98	98	96	98	96

Cuadro 63: Autómatas que pasaron la prueba con el generador 43369

# AC	P_1	P_2	P_3	P_4	P_5
22889	97	96	96	96	98
23209	96	96	97	96	97
27285	97	97	97	98	96
39334	97	98	96	96	96

Cuadro 64: Autómatas que pasaron la prueba con el generador 43370

# AC	P_1	P_2	P_3	P_4	P_5
4590	96	96	96	96	98
38245	96	98	98	99	97
39334	98	96	96	96	96

Cuadro 65: Autómatas que pasaron la prueba con el generador 57630

BIBLIOGRAFÍA

[1] **ALCOVER GARAU Pedro María, GARCÍA CARRASCO José M. y HERNÁNDEZ ENCINAS, Luis,** Novática, vol 174 pags 59 - 65, marzo-abril 2005.

[2] **BAO, F.** Cryptanalysis of a partially known cellular automata cryptosystem, IEEE Transactions on Computers, volúmen 53, número 11, pags. 1493-1497, 2004.

[3] **BENKINIOUAR, Moad y BENMOHAMED, Mohamed**. Cellular automata for Cryptography, IEEE Transactions on Computers, 2004.

[4] **DIFFIE, W., Y HELLMAN, M. E.** New directions in cryptography. IEEE Transactions on Information Theory IT-26, 6 (Nov. 1976), 644-654.

[5] **FUENTE, S. Y GONZÁLEZ, L**. Autómatas Celulares, Pisuerga, 2000.
URL http://pisuerga.inf.ubu.es/cgosorio/AleF/AutomatasCelulares.pdf

[6] **FÚSTER, A., DE LA GUÍA, D., HERNÁNDEZ, L., MONTOYA, F. Y MUÑOZ J.** Técnicas Criptográficas de Protección de Datos. RA-MA, 2000.

[7] **GUAN, P.** Cellular automaton public-key cryptosystem, Complex Systems, volumen 1, pags. 51-57", 1987.

[8] **HOYA WHITE**, Sara. Aplicaciones de los Autómatas Celulares a los Criptosistemas de Cifrado en Flujo, Universidad de Salamanca, 2002.
URL http://www.criptored.upm.es/descarga/GradoSaraHoyaWhite.zip

[9] **LÓPEZ LUCENA, Manuel José**, Criptografía y Seguridad en Computadores, Tercera Edicíon (Versión 1.00) Junio de 2001.

[10] **MEIER, W. Y STAFFELBACH, O.** Analysis of pseudo random sequences generated by cellular automata, Advances in Cryptology - EUROCRYPT 91, Lecture Notes in Computer Science, volumen 547, pags 186-189, 1992.

[11] **NANDI, S. ; KAR, B.K. Y CHAUDHURI, Pal**, Theory and applications of cellular automata in cryptography, IEEE Transactions on Computers, volumen 43, pags 1346-1357, 1994.

[12] **U.S. DEPARTMENT OF COMMERCE**, National Institute of Standards and Technology, Security Requirements for Cryptographics Modules FIPS 140-1, 1994.

[13] - - - - - - - -, National Institute of Standards and Technology, Security Requirements for Cryptographics Modules FIPS 140-2, 2001.

[14] **WOLFRAM, Stephen**, A New Kind of Science, Wolfram Media Inc, 2002.

[15] **WOLFRAM, Stephen**, Cryptography with cellular automata, Advances in Cryptology: Crypto '85 Proceedings, LNCS, vol. 218, pp 429-432, 1986.