INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Trabajo Terminal

“Caracterización de un algoritmo criptográfico a través de un autómata celular.”

2013-B021

Presentan

Erick Eduardo Aguilar Hernández

Jessica Matías Blancas

Directores

M. en C. Nidia Asunción Cortez Duarte

Dr. Genaro Juárez Martínez

Febrero 2015

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPTO
SUBDIRECCIÓN ACADÉMICA

No. de TT: 2013-B02 26-Febrero-2015

Documento Técnico

"Caracterización de un algoritmo criptográfico a través de un autómata celular"

Presentan

Erick Eduardo Aguilar Hernández[1]

Jessica Matías Blancas[2]

Directores

M. en C. Nidia Asunción Cortez Duarte **Dr. Genaro Juárez Martínez**

Resumen

En el presente reporte se presenta la documentación técnica del Trabajo Terminal 2013-B021, titulado: "Caracterización de un algoritmo criptográfico a través de un autómata celular", el cual incluye la elaboración de un estudio comparativo de un conjunto de algoritmos criptográficos con el objeto de seleccionar uno para ser implementado en su forma estándar y a través de un autómata celular, comparando el comportamiento de ambas implementaciones.

Palabras clave: autómata celular, criptografía, sistema dinámico discreto.

[1] isc.ErickAguilar@gmail.com
[2] iscmatias@outlook.com

ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADEMICA

DEPARTAMENTO DE FORMACIÓN INTEGRAL E INSTITUCIONAL

COMISION ACADÉMICA DE TRABAJO TERMINAL

México, D.F. a 4 de marzo de 2015

DR. FLAVIO ARTURO SÁNCHEZ GARFIAS
PRESIDENTE DE LA COMISIÓN ACADÉMICA
DE TRABAJO TERMINAL
P R E S E N T E

Por medio del presente, se informa que los alumnos que integran el TRABAJO TERMINAL: 2013-B021, titulado "Caracterización de un algoritmo criptográfico a través de un autómata celular" concluyeron satisfactoriamente su trabajo.

Los discos (DVDs) fueron revisados ampliamente por sus servidores y corregidos, cubriendo el alcance y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la Comisión que Usted preside.

ATENTAMENTE

Dr. Genaro Juárez Martínez

ATENTAMENTE

M. en C. Nidia Asunción Cortez Duarte

Advertencia

“Este documento contiene información desarrollada por la Escuela superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan.”

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Índice

Índice de cuadros

Índice de figuras

1. Introdución

El intercambio de información digital generalmente se realiza a través de canales inseguros y como consecuencia la información podría verse comprometida es por eso que se requiere proveer de servicios de seguridad que basan sus principios en primitivas criptográficas. Las primitivas criptográficas cifrado/descifrado y firma/verificación hacen uso de algoritmos criptográficos considerados seguros computacionalmente.

Por otra parte los Autómatas Celulares (AC) son sistemas dinámicos discretos que han sido estudiados como máquinas capaces de efectuar computación. Existen, inclusive, autómatas celulares que pueden ser configurados para efectuar cálculo de propósito universal. Los AC son capaces de generar comportamientos aplicables a la criptografía, existen varios motivos por los cuales resultan atractivos como mecanismos de cifrado, en principio porque la computación se realiza en paralelo, la programación de las funciones de transición es simple.

El problema de diseñar autómatas para propósitos específicos como lo es el cifrado de datos, es conocer la función de transición del AC asociada al comportamiento a utilizar, para realizar dicha tarea lo primero que se debe hacer es caracterizar la dinámica abstrayendo sus cambios a lo largo del tiempo.

En la actualidad los avances que conjuntan ambas áreas de las ciencias de la computación resultan incipientes, sin embargo el potencial que poseen los autómatas celulares para realizar cálculos en aritmética de campos finitos en especifico los campos de Galois, los vuelven aplicables a la criptografía, puesto que dichos campos son la base sobre los cuales se construyen los algoritmos criptográficos modernos.

En el presente trabajo terminal se propone examinar un conjunto de algoritmos criptográficos comparándolos con el objetivo de seleccionar uno, el algoritmo seleccionado será caracterizado en función de sus operaciones primitivas y de esta manera obtener la función de transición asociada al autómata que las describa, para poder llevar a cabo su implementación en forma de un AC, posteriormente se comparará el AC, con respecto a la implementación estándar del algoritmo criptográfico para observar su comportamiento.

1.1. Objetivo

Implementar un autómata celular que caracterice a un algoritmo criptográfico para comparar su comportamiento respecto al comportamiento tradicional.

1.1.1. Objetivos específicos

- Realizar un estudio comparativo de los algoritmos criptográficos y su factibilidad de implementación en forma de un autómata celular.
- Obtener las funciones de transición del autómata con base a la caracterización del algoritmo seleccionado.
- Implementar el autómata celular de la función de transición obtenida y el algoritmo criptográfico seleccionado en su forma estándar.

Recomendación

Para poder comprender el contenido del presente trabajo es necesario tener conocimiento de los siguientes temas:

- **Teoría de autómatas**: conceptos de computación, autómatas y procesos computables.
- **Teoría de autómatas célulares**: definición y tipos de vecindades, clasifiación de Wolfram, funciones de transición exactas, funciones de transición definidas por ecuaciones totalísticas y semí-totalísticas.
- **Teoría de la complejidad**: orden de complejidad y notación asintótica.
- **Matemática Discreta**: Divisibilidad entera y relaciones de congruencia, anillos enteros, álgebra de Boole, teoría de conjuntos, relaciones, funciones entre conjuntos y diagramas conmutativos.
- **Criptografía**: conceptos básicos, primitivas criptográficas, criptagrafía clásica, criptografía de clave publica, criptografía de clave privada y modos de operación.
- **Álgebra Lineal**: teoría de matrices, espacios vectoriales, bases y transformaciones lineales.
- **Álgebra Moderna**: teoría general de campos finitos en especial de los campos de Galois binarios, anillos de polinomios con entradas modulares, divisibilidad y congruencia en anillos polinomiales, reductibilidad polinomial, logaritmos y antilogaritmos discretos.

Marco Teórico

2. Marco Teórico

2.1. Criptografía

La criptografía se define como la ciencia que se dedica al estudio de la escritura secreta, es decir, estudia los mensajes que sometidos a un proceso de transformación, se convierten en difíciles o imposibles de leer. [1]. La criptología proviene de la unión de los términos griegos kripto (oculto) y logos (estudio).

La criptología está dividida en los siguientes campos.

- **Esteganografía:** se ocupa de esconder mensajes con información privada, enviados por un canal inseguro, de forma que el mensaje sea desapercibido. Regularmente el mensaje se oculta dentro de datos con formatos de video, imágenes, audio o mensajes de texto [2].
- **Estegoanálisis:** es la contraparte de la estenografía, la cual dedica su atencion a de detectar mensajes ocultos con técnicas esteganográficas, para así reducir o eliminar la seguridad que aporta la esteganografía.
- **Criptografía:** es el estudio de técnicas matemáticas relacionadas con aspectos de seguridad de la información, como la confidencialidad, la integridad de los datos, autenticación, y el origen de datos [3]. Los objetivos que tiene la seguridad de la información son los siguientes: la intimidad o confidencialidad, la integridad de datos, la autenticación, y el no repudio, cuyos elementos teóricos son la base para el desarrollo de nuestro trabajo terminal.
- **Criptoanálisis:** es el estudio de técnicas matemáticas que se oponen a las técnicas criptográficas, y en una idea más en general, comprometer los servicios de seguridad de la información [3]. Su objetivo es buscar el punto débil de las técnicas criptográficas para explotarlas y así eliminar o reducir la seguridad que teóricamente aporta la criptografía, al intento de criptoanálisis se le llama ataque.

2.2. Clasificación de la criptografía

A través de la historia se desarrollaron varias técnicas para que ciertos mensajes solo pudieran ser interpretados por los usuarios autorizados. Al conjunto de técnicas que no es necesario computarizar ni digitalizar se les conoce como criptografía clásica algunos ejemplos de esta criptografía son: affin, Vigenieré, Hill, entre otros.

Sin embargo fué hasta la segunda guerra mundial, cuando un ingeniero alemán Edward Hebern, dió a conocer una máquina llamada enigma. El gobierno Alemán adquirió esta máquina para emplearla en la transmisión de sus mensajes, creían que un criptograma de enigma era indescifrable y debido a las necesidades a nivel gubernamental para cifrar información surgen propuestas de algoritmos que cubran dicho propósito, mismas que se clasifican de acuerdo a las llaves que usan.

2.2.1. Cifradores asimétricos ó de clave pública

Los algoritmos de clave pública introducidos por Whitfield Diffie y Martin Hellman a mediados de los años 70, exhiben una novedad fundamental y esta es que las claves no son únicas, sino que forman pares, introduciendo el concepto de criptografía de clave pública. Con las claves públicas no se requiere que el remitente y el destinatario se pongan de acuerdo previamente sobre la clave que se empleará. Todo lo que se necesita es que, antes de iniciar la

comunicación secreta, el remitente consiga una copia de la clave pública del destinatario. Los algoritmos asimétricos emplean generalmente longitudes de claves mayores que los simétricos, se consideran claves seguras de al menos 1024 bits, excluyendo aquellos basados en curvas elípticas. Además, la complejidad de cálculo de estos últimos los hace considerablemente más lentos que los algoritmos de cifrado simétricos. Entre los cifradores simétricos más populares se encuentran:

- RC4.
- RSA.
- DeffieHellman.
- Criptografia de curvas elípticas (ECC).

2.2.2. Cifradores simétricos ó de clave privada

Son quellos en los cuaesl se usa una misma clave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez que ambas partes tienen acceso a esta clave, el remitente cifra un mensaje usando la clave, lo envía al destinatario, y éste lo descifra con la misma clave.

Cifradores de flujo

Los cifradores de flujo son aquellos en los que los algoritmos de cifrado, parten del texto claro convirtiéndolo en texto cifrado bit a bit. Esto se logra construyendo un generador de flujo de clave. Un flujo de clave es una sucesión de bits en el cual el tamaño es arbitrario, se utiliza para encubrir el contenido de un flujo de datos, combinando el flujo de clave con el flujo de datos mediante la función XOR. Si el flujo de clave es seguro, el flujo de datos cifrados también lo será.

Cifradores de bloques

Una gran parte de los algoritmos de cifrado simétrico operan dividiendo el mensaje que se pretende codificar en bloques de tamaño fijo, y aplican sobre cada uno de ellos una combinación mas o menos complicada de operaciones de sustitución, difusión e incluso transposiciones. A Estos algoritmos se les denomina, cifrados por bloques.

Algoritmo DES

El Data Encryption Standard (DES) es el más conocido cifrador de bloques además de ser un algoritmo simétrico. La idea básica de un sistema de cifrado de producto como lo es el algoritmo DES es construir una función de cifrado compleja mediante la composición de varias operaciones simples, que ofrecen individualmente protección básica sus operaciones. El algoritmo DES codifica bloques de 64 bits empleando claves de 56 bits.

La descripción del algoritmo consta de 16 fases idénticas de proceso, denominadas rondas. También hay una permutación inicial y final denominada P_i y P_f, que son funciones inversas entre sí (P_i es la función contraria a la acción de P_f recíprocamente). Antes de comenzar con las rondas, el bloque es dividido en dos mitades de 32 bits y procesadas alternativamente. A este esquema de cruce se llama función de Feistel. Una vez que el mensaje se ha dividido en

dos mitades es cuando se introducen a la primera ronda que consta de cuatro pasos que a continuación se explican:

1 *Expansión:* la mitad del bloque de 32 bits se expande a 48 bits mediante la permutación de expansión, duplicando algunos de los bits.

2 *Mezcla:* el resultado se combina con una subclave utilizando una operación XOR. Se obtendrán dieciséis subclaves, una para cada ronda, las claves siguientes se derivan de la clave inicial mediante la generación de subclaves descrita más abajo.

3 *Sustitución:* tras mezclarlo con la subclave, el bloque es dividido en ocho trozos de 6 bits antes de ser procesados por las S-box, o cajas de sustitución. Cada una de las ocho S-box reemplaza sus seis bits de entrada con cuatro bits de salida, de acuerdo con una trasformación no lineal, especificada por una tabla de búsqueda. Las S-box constituyen el núcleo de la seguridad de DES sin ellas, el cifrado sería lineal, y fácil de romper.

4 *Permutación:* finalmente, las 32 salidas de las S-box se reordenan de acuerdo a una permutación fija; la P-box.

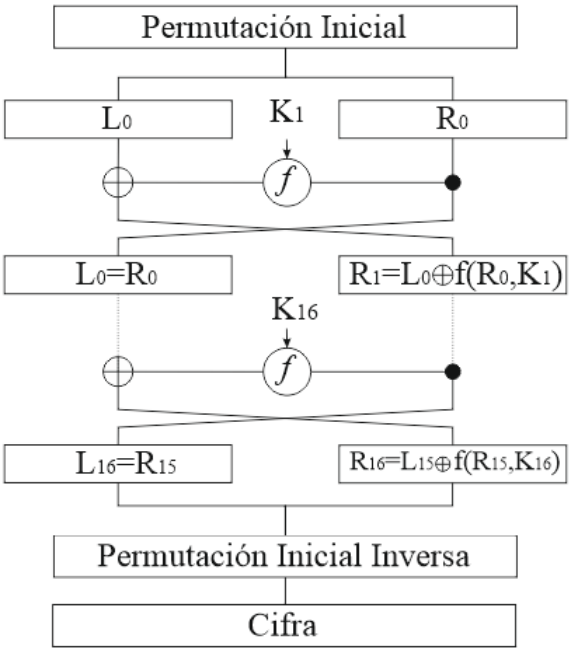

Figura 1: Diagrama de bloques del cifrado DES.

Algoritmo AES

En 1997, el National Institute for Standards and Technology(NIST) decidió realizar un concurso para seleccionar un nuevo algoritmo de cifrado que fuera capaz de proteger información sensible durante el siglo XXI. Este algoritmo se denominó Advanced Encryption Standard (AES). El 2 de enero de 1997 el NIST comunicó su propósito de desarrollar AES, con la ayuda de la industria y de la comunidad criptográfica y el 12 de septiembre hizo la convocatoria formal. En esta convocatoria se indicaban varias condiciones para los algoritmos que se presentaran:

- Ser de dominio público, disponible para todo el mundo.
- Ser un algoritmo de cifrado simétrico y soportar bloques de, como mínimo, 128 bits.

- Las claves de cifrado podrían ser de 128, 192 y 256 bits.
- Ser implementable tanto en hardware como en software.

El 20 de agosto de 1998 el NIST anunció los 15 algoritmos admitidos en la primera conferencia AES, entre los cuales se encontraba el algoritmo Rijndael, quien finalmente fue el ganador.

AES es un sistema de cifrado por bloques, diseñado para manejar longitudes de clave, AES tiene un tamaño de bloque fijo de 128 bits y tamaños de clave de 128, 192 o 256 bits. Realiza varias de sus operaciones internas a nivel de bits, interpretando estos como elementos de un campo de Galois GF(2^8).

AES, a diferencia de algoritmos como DES, no posee estructura de red de Feistel. En su lugar se ha definido cada ronda como una composición de cuatro funciones invertibles diferentes, formando tres capas, diseñadas para proporcionar resistencia frente a criptoanálisis. AES opera en una matriz de $4*4$ bytes, llamada state. Cada una de las funciones tiene un propósito preciso, la capa de mezcla lineal que se encuentran presente en las funciones *ShiftRows* y *MixColumns* permite obtener un alto nivel de difusión a lo largo de varias rondas. La capa función no lineal que se encuentra presente en la función *SubByte* la cual consiste en la aplicación paralela de S-Box. Y por último la capa de adición de clave es un simple Or-Exclusivo entre el estado intermedio y la subclave correspondiente en cada ronda. A continuación se explica brevemente el modo de operación de cada una de las funciones del algoritmo AES.

La primera etapa es la de **AddRoudKey** en esta etapa se hace la operación Or-Exclusivo, entre la clave inicial y el texto claro.

Después de realizar la etapa de AddRoudKey procede la etapa **SubBytes**, cada byte en la matriz es actualizado usando la **S-box** de Rijndael de 8 bits. La S-box utilizada proviene de la función inversa alrededor del GF(2^8), y es reconocida por tener grandes propiedades de no linealidad. Esta propiedad evita ataques que están basados en propiedades algebraicas.

El siguiente paso es la función **ShiftRows** la cual opera en las filas del state, esta función rota de manera cíclica los bytes en cada fila por un determinado offset. En AES, la primera fila queda en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta filas son rotadas por los offsets de dos y tres respectivamente.

En la etapa de **MixColumns**, los cuatro bytes de cada columna del state se combinan usando una transformación lineal columna por columna. La función MixColumns toma cuatro bytes como entrada y devuelve cuatro bytes, donde cada byte de entrada influye todas las salidas de cuatro bytes. Junto con ShiftRows, MixColumns implica difusión en el cifrado. Cada columna se trata como un polinomio GF(2^8) y luego se multiplica el módulo $x^4 + 1$ con un polinomio fijo $a(x)$.

2.3. Aritmética modular

Este tipo de aritmética pertenece a una rama de las matemáticas que es extremadamente útil en Criptografía, ya que permite realizar cálculos complejos, manteniendo siempre una representación numérica compacta definida, puesto que solo maneja un conjunto finito de números enteros.

La aritmética modular se fundamenta en una relación de congruencia entre los enteros y se denota con $a \ mod \ n$. Una relación de congruencia entre enteros es compatible con las ope-

raciones en el anillo de enteros: suma, resta, y multiplicación. Para un determinado módulo n, la relación de congruencia se define de la siguiente manera:

DEFINICIÓN 1 (RELACIÓN DE CONGRUENCIA)
Sean a y b dos números que se encuentran en la misma clase de congruencia módulo n, si ambos dejan el mismo residuo si los dividimos entre n, o, equivalentemente, si $a - b$ es un múltiplo de n.

Dados tres números $a, b, n \epsilon \mathbb{N}$ decimos que a es congruente con b módulo n, y se puede expresar cómodamente utilizando la notación de Gauss [4]:

$$a \equiv b \ (mod\ n)$$

si se cumple:

$$a = b + kn \text{ para algún } k \epsilon \mathbb{Z}$$

Por ejemplo:

$$63 \equiv 83 \ (mod\ 10)$$

ya que ambos 63 y 83 dejan el mismo resto (3) al dividir entre 10, o, equivalentemente, $63 - 83$ es un múltiplo de 10. Se lee: *63 es congruente con 83, módulo 10.*

Ahora podemos definir las operaciones de suma y producto en este tipo de conjuntos:

DEFINICIÓN 2 (OPERACIONES EN UNA RELACIÓN DE CONGRUENCIA)

- $a + b \equiv c \ (mod\ n) \iff a + b = c + kn; \quad k \epsilon \mathbb{Z}$
- $ab \equiv c \ (mod\ n) \iff ab = c + kn; \quad k \epsilon \mathbb{Z}$

Propiedades de la suma y producto en una relación de congruencia.

Propiedades de la suma:

- *Asociativa:* para todo $a, b,$ y $c \epsilon \mathbb{Z}_n \quad (a + b) + c \equiv a + (b + c) \ (mod\ n)$
- *Conmutativa:* para todo a y $b \epsilon \mathbb{Z}_n \quad a + b \equiv b + a \ (mod\ n)$
- *Elemento Neutro:* para todo $a \epsilon \mathbb{Z}_n \ \exists\ 0$ tal que $\ a + 0 \equiv a \ (mod\ n)$
- *Elemento Simétrico (opuesto):* para todo $a \epsilon \mathbb{Z}_n \ \exists\ b$ tal que $a + b \equiv 0 \ (mod\ n)$

Propiedades del producto:

- *Asociativa:* para todo $a, b,$ y $c \epsilon \mathbb{Z}_n \ (a \bullet b) \bullet c \equiv a \bullet (b \bullet c) \ (mod\ n)$
- *Conmutativa:* para todo a y $b \epsilon \mathbb{Z}_n \quad a \bullet b \equiv b \bullet a \ (mod\ n)$
- *Elemento Neutro:* para todo $a \epsilon \mathbb{Z}_n \quad \exists\ 1$ tal que $\equiv a \bullet 1 \equiv a \ (mod\ n)$

Propiedades del producto con respecto de la suma.

- *Distibutiva:* para todo $a, b,$ y $c \epsilon \mathbb{Z}_n \ (a + b) \bullet c \equiv (a \bullet c) + (b \bullet c) \ (mod\ n)$

Un punto importante en la aritmética modular, es el tema de en qué circunstancias una congruencia lineal puede ser resuelta, y esta se describe mediante el teorema de congruencia lineal que se señala a continuación. Si a y b dos números enteros cualesquiera y n es un número entero positivo, entonces la congruencia tiene una solución para x si y sólo si b es divisible por el máximo común divisor d de a y n (denotado mediante mcd(a, n)).

$$a \equiv b(mod\ n) \tag{1}$$

Cuando éste sea el caso, y x_0 es una solución de (1), entonces el conjunto de todas las soluciones está denotado por:

$$x_0 + k\frac{n}{d} |\ k\ \epsilon\ \mathbb{Z}$$

En particular, existirán exactamente $d = mcd(a, n)$ soluciones en el conjunto de residuos: $\{0, 1, 2, ..., n-1\}$.

2.4. Teoría de campos finitos

Uno de los pilares sobre los que se sustenta la criptografía moderna es la matemática, en especial la matemática discreta, teoría de números, álgebra moderna, el álgebra lineal, etc. Dentro de las teorías matemáticas que han servido de impulso para las ciencias de la computación se encuentra la teoría de campos finitos, cuyas aplicaciones van desde la teoría de la información (desarrollada por Claude Shannon) hasta el desarrollo de algoritmos criptograficos modernos. La teoría de campos finitos resulta aplicable a la criptografía ya que permite extender las funciones afines a nuevas estructuras algebraicas más complejas y con propiedades interesantes que establecen su estructuras y operaciones permitidas en sistemas de axiomas, por lo que si se requiere ahondar en temas de criptografía moderna es necesario conocer y entender la teoría de campos finitos, en especial los campos de Galois y los axiomas sobre los que estos se definen.

Un axioma es un fundamento de verdad o un hecho que se acepta por verdadero pero no se demuestra. Los axiomas que fundamentan la matemática son pocos, esencialmente estos son los de Peano, que están basados en razonamientos que de alguna manera, su validez es evidente, dicho de otra manera, se aceptan como verdaderos.

Un sistema de axiomas definen una estructura algebraica bajo una ó más operaciones binarias definidas y cerradas sobre un conjunto no vació. Un caso particular es el conjunto de los números enteros, denotado por $\mathbb{Z} = \{...-2, -1-, 0, 1, 2, ...\}$, en el cual se definen dos operaciones binarias la suma de enteros y producto de enteros, ambas operaciones básicas de la aritmética elemental de números enteros sin embargo las operaciones cumplen una serie de axiomas de tal forma que $\mathbb{Z}$ conforma una estructura algebraica denominada anillo.

Es posible que conjuntos formados por elementos de diferente naturaleza como números, matrices, polinomios, etc. Que están provistos de distintas operaciones tengan, sin embargo el mismo comportamiento algebraico. Esto se debe a que varios de estos conjuntos conforman la misma estructura aunque sus operaciones correspondientes pueden llegar a ser muy diferentes entre si. Por lo que es necesario definir los conceptos de operación y establecer los axiomas que dichas operaciones deben cumplir para poder clasificar las estructuras que forman. [5]

Cuando hablamos de una operación binaria nos tenemos que referir a entes matemáticos dónde podemos realizar las operaciones comunes.

DEFINICIÓN 3 (OPERACIÓN BINARIA)
Sea S un conjunto no vacío, se define la función $*$ talque $* : S\text{x}S \rightarrow S$, $*$ es llamada operación binaria sobre S que cumple la propiedad de cerradura. Cabe mencionar que existen operaciones sobre conjuntos que no son cerradas. [5]

Una de las estructuras algebraicas más simples es la llamada estructura de **grupo**

DEFINICIÓN 4 (GRUPO)
Sea G un conjunto no vació y la operación binaria $*$ sobre G, se dice que G es grupo (G,$*$) si $*$ cumple los axiomas de grupo.

G-1 Asociatividad $\forall$ a,b,c $\in$ G, a$*$(b$*$c)=(a$*$b)$*$c

G-2 Existencia de elemento neutro, $\exists$ e $\in$ G tal que $\forall$ a $\in$ G, a$*$e=e$*$a=a

G-3 Existencia de inverso, $\forall$ a $\in$ G $\exists$ un elemento $a^{-1} \in$ G tal que a$*$ a^{-1} = $a^{-1}$$*$a=e.

G-4 Conmutatividad, si además $\forall$ a,b $\in$ G, se cumple que, a$*$b=b$*$a entonces el grupo es llamado **Grupo Abeliano ó Grupo Conmutativo** [6].

Ejemplos de grupos:

- Las posibles manipulaciones del cubo de rubik forman un grupo.
- La suma de matrices sobre un campo $\mathbb{K}$. ($M_{nxm}(\mathbb{K})$,+).
- El conjunto de polinomios con variables x y coeficientes en $\mathbb{N}$. ($P(x)_{\mathbb{N}}$,+).

En diversos conjuntos de elementos existe más de una operación binaria cerrada, por ejemplo en el conjunto de números enteros $\mathbb{Z}$ hay dos operaciones cerradas, la suma y el producto. Por lo que es necesario definir otro tipo de estructura algebraica conocida como anillo.

DEFINICIÓN 5 (ANILLO)
Sea R un conjunto no vacío, con dos operaciones binarias $*$ y $\circ$, se dice que R es un anillo (R,$*$,$\circ$), si $*$ y $\circ$ cumplen los axiomas de anillo:

A-1 (R,$*$) es grupo abeliano.

A-2 Asociatividad del operador $\circ$, $\forall$ a,b,c $\in$ R, a$\circ$(b$\circ$c)=(a$\circ$b)$\circ$c.

A-3 El operador $\circ$ se distribuye respecto a $*$, es decir, $\forall$ a,b,c $\in$ R se cumple que a$\circ$(b$*$c)=a$\circ$b$*$a$\circ$c y (b$*$c)$\circ$a=b$\circ$a$*$c$\circ$a. Si además se cumple que:

A-4 Conmutatividad del operador $\circ$, $\forall$ a,b $\in$ R, a$\circ$b=b$\circ$a Entonces el anillo R es llamado **anillo conmutativo**.

A-5 Existencia de elemento neutro, $\exists$ e $\in$ G tal que $\forall$ a $\in$ G, a$\circ$e=e$\circ$a=a. Entonces el anillo R es llamado **anillo unitario**, si también cumple (4 - $\circ$) el anillo R es llamado **anillo unitario conmutativo**.

A-6 Dominio entero, $\forall$ a,b $\in$ R, a$\circ$b$\neq$ 0 $\Leftrightarrow$ a$\neq$0 ó b$\neq$0. Entonces el anillo R es llamado **dominio entero** [6].

Ejemplos de Anillos:

- El conjunto de enteros módulo n, con n $\in \mathbb{N}$, ($\mathbb{Z}_n$,+,.) es anillo conmutitativo unitario.

- El conjunto de matrices con entradas en $\mathbb{K}$. ($M_{nxm}(\mathbb{K})$,+,.) Es anillo unitario.
- El conjunto de polinomios con variables x y coeficientes en $\mathbb{Z}$. ($P(x)_{\mathbb{Z}}$,+) es anillo conmutitativo unitario.

DEFINICIÓN 6 (CAMPO)
Sea F un conjunto no vacío, con dos operaciones binarias, la suma $+$ y el producto $\bullet$, se dice que F es campo $\mathbb{F}$, si $+$ y $\bullet$ cumplen los axiomas de campo:

F1 (F,$+$) es grupo abeliano y se denomina **grupo aditivo del campo**.

F2 Existe un elemento **0** $\in$ F tal que (F$\backslash\{$**0**$\}$,$\bullet$) es grupo abeliano y se denomina **grupo multiplicativo del campo**.

F3 El producto $\bullet$ se distribuye respecto a la suma $+$, es decir, $\forall$ a,b,c $\in$ F se cumple que: a$\bullet$(b$+$c)=a$\bullet$b$+$a$\bullet$c y (b$+$c)$\bullet$a=b$\bullet$a$+$c$\bullet$a. Si además:

F4 F $\subset \mathbb{I}_p \bigcup \{0\}$, Es decir F son los primeros p números naturales con p primo, F se denomina **campo finito de oren p** y se denota por $\mathbb{F}_p$, se dice que p es la característica del campo. [6]

Teoremas: Un campo finito $\mathbb{F}_p$ satisface lo siguiente.

1. $\mathbb{F}_p = \mathbb{Z}_{/p} = \mathbb{Z}_{/p\mathbb{Z}}$ es decir $\mathbb{F}_p$ es el conjunto de clases residuales módulo p.
2. $\forall$ a $\in \mathbb{F}_p$ $\exists$ e $\in \mathbb{F}_p$ elemento distinguido y un $n \in \mathbb{N}$ tal que $a = e^n$, e se denomina generador de $\mathbb{F}_p$. Al Conjunto de elementos generadores $\mathbb{F}_p^*$ de $\mathbb{F}_p$ se denomina conjunto primitivo y al conjunto generador por e se le denomina **grupo cíclico**.
3. $\mathbb{F}$ es un espacio vectorial sobre $\mathbb{F}_p$ con base $\mathbb{F}_p^*$.
4. Dado un elemento primitivo e $\in \mathbb{F}_p$, decimos que el logaritmo ó indice de un elemento a $\in \mathbb{F}_p^*$ es el menor entero no negativo que cumple $a = e^n$.

Consideremos el anillo $\mathbb{Z}_2$ con suma y producto definidos como sigue.

+	**0**	**1**
0	0	1
1	1	0

Cuadro 1: Tabla de la suma en $\mathbb{Z}_2$.

$\bullet$	**0**	**1**
0	0	0
1	0	1

Cuadro 2: Tabla de la multiplicación en $\mathbb{Z}_2$.

Es fácil ver que $\mathbb{Z}_2$ es campo finito y que la suma corresponde con la compuerta lógica XOR y el producto con la compuerta lógica AND.

DEFINICIÓN 7 (POLINOMIOS CON COEFICIENTES EN $\mathbb{Z}_n$)
Sea $\mathbb{Z}_n$ un anillo arbitrario. Un polinomio sobre $\mathbb{Z}_n$ es una expresión de la forma. [4]

$$f(x) = \sum_{i=0}^{k} a_i x^i = a_0 + a_0 x + ... + a_n x^n \tag{2}$$

Donde n es un entero no negativo, los coeficientes $a_i i = 0, .., n$, son elementos de $\mathbb{Z}_n$.

Sean $f(x) = \sum_{i=0}^{k_1} a_i x^i$ y $g(x) = \sum_{i=0}^{k_2} b_i x^i$ polinomios sobre $\mathbb{Z}_n$ las operaciones entre polinomios definen como sigue.

- $f(x)$ + $g(x)$=$\sum_{i=0}^{max\{k_1,k_2\}}(a_i + b_i$ mod n$)x^i$.
- $f(x)\bullet g(x)$=$\sum_{i=0}^{k_1+k_2}(a_i \bullet b_i$ mod n$)x^i$

El conjunto formado por los polinomios sobre $\mathbb{Z}_n$ con las operaciones mencionadas forma un anillo y se le denomina anillo de polinomios en $\mathbb{Z}_n$ y se denota por $\mathbb{Z}_n$[x]. El elemento 0 de $\mathbb{Z}_n$[x] es el polinomio con todos su coeficientes 0, y se le conoce como el polinomio cero, al polinomio con coeficientes cero a excepción del coeficiente en x^0 es decir el 1 polinomio, ambos polinomios son constantes. [6] Consideremos los polinomios $f(x) = x^2 + 1$ y $g(x) = x^5 + x^3 + x^2 + 1$ en $\mathbb{Z}_2[x]$.

$$\begin{aligned} f(x) + g(x) &= (x^2+1) + (x^5 + x^3 + x^2 + 1) \\ &= (x^5) + (x^3) + (x^2 + x^2) + (x^0 + x^0) \text{ agrupando términos semejantes} \\ &= (x^5) + (x^3) + (1 \oplus 1)x^2 + (1 \oplus 1)x^0 \text{ según la tabla de la suma en } \mathbb{Z}_2 \\ (f+g)(x) &= x^5 + x^3 \end{aligned}$$

$$\begin{aligned} f(x) \bullet g(x) &= (x^2+1)(x^5 + x^3 + x^2 + 1) \\ &= (x^2)(x^5 + x^3 + x^2 + 1) + (1)(x^5 + x^3 + x^2 + 1) \\ &= (x^7 + x^5 + x^4 + x^2) + (x^5 + x^3 + x^2 + 1) \\ &= (x^7) + (x^5 + x^5) + (x^4) + (x^3) + (x^2 + x^2) + (1+1)x^0 \\ &= (x^7) + (1 \oplus 1)x^5 + (x^4) + (x^3) + (1 \oplus 1)x^2 + (1 \oplus 1)x^0 \\ &= (x^7) + (0)x^5 + (x^4) + (x^3) + (0)x^2 + (0)x^0 \\ (fg)(x) &= x^7 + x^4 + x^3 \end{aligned}$$

Resultados importantes: el anillo $\mathbb{Z}_n$[x] hereda las propiedades del anillo $\mathbb{Z}_n$.

- $\mathbb{Z}_n[x]$ es anillo conmutativo unitario y dominio entero si y solo si $\mathbb{Z}_n$ es campo, $\mathbb{Z}_n$[x] no es campo ya que lo únicos elementos con inverso multiplicativo son los polinomios constantes y no nulos.
- El algoritmo de la división válido en $\mathbb{Z}_n$ también es valido para $\mathbb{Z}_n$[x], es decir sean polinomios $f(x)$ y $g(x)$ con coeficientes en $\mathbb{Z}_n$, existen polinomios únicos $q(x)$ y $r(x)$ en $\mathbb{Z}_n$[x] tales que: $f(x) = q(x)g(x) + r(x)$.
- Dados dos polinomios $f(x), g(x) \in \mathbb{Z}_n$[x] existe un polinomio d(x) $\in \mathbb{Z}_n$[x] tal que d(x) divide $f(x)$ y $g(x)$ y se le conoce como máximo común divisor de $f(x)$, $g(x)$.
- Un polinomio $f(x)$ $\mathbb{Z}_n$[x] es reducible si existen polinomios $h(x)$, $c(x) \in \mathbb{Z}_n$[x] con grado mayor ó igual a 1 tales que $f(x) = g(x)c(x)$. En caso contrario se dice que a(x) es irreducible.
- Dos polinomios $f(x), g(x) \in \mathbb{Z}_n$ se dice que son congruentes módulo un tercer polinomio m(x) $\in \mathbb{Z}_n$ irreducible y no cero si m $|(f-g)$ y se escribe $f(x)$ = $g(x)$ mod $m(x)$.

- Sea m(x) $\in \mathbb{Z}_n$[x] irreducible, el conjunto cociente $\mathbb{Z}_n$[x]/m(x) conformado por las clases de equivalencia de los elementos $\mathbb{Z}_n$[x] módulo m(x) es campo, el polinomio m(x) es conocido como polinomio de reducción. [6]

Consideremos el polinomio irreducible m(x) $\in \mathbb{Z}_2$[x], m(x)=$x^2 + x + 1$ entonces el conjunto cociente $\mathbb{Z}_2$[x]/m(x) queda definida como sigue.

+	[0]	[1]	[x]	[x+1]
[0]	[0]	[1]	[x]	[x+1]
[1]	[1]	[0]	[x+1]	[x]
[x]	[x]	[x+1]	[0]	[1]
[x+1]	[x+1]	[x]	[1]	[0]

Cuadro 3: Tabla de la suma en $\mathbb{Z}_2$[x]/m(x).

•	[0]	[1]	[x]	[x+1]
[0]	[0]	[0]	[0]	[0]
[1]	[0]	[1]	[x]	[x+1]
[x]	[0]	[x]	[x+1]	[1]
[x+1]	[0]	[x+1]	[1]	[x+1]

Cuadro 4: Tabla de la multiplicación en $\mathbb{Z}_2$[x]/m(x).

2.4.1. Campos de Galois binarios

DEFINICIÓN 8 (CAMPO DE GALOIS)
Existe un campo de orden q si y solo q es potencia de un número primo p, es decir $q = p^m$ con $m \in \mathbb{N}$ dicho campo se denomina campo de Galois denotado GF(p^m) con característica q y grado de extensión m [6].

Campos de Galois binarios

Un campo de Galois binario GF(2^m) es un campo finito de característica 2 y extensión m. GF(2^m) es espacio vectorial sobre GF(2) por consiguiente existen m elementos $\alpha_0, \alpha_1 ..., \alpha_{m-1}$ en GF(2^m) tales que cada elemento $\alpha \in$ GF(2^m) puede ser descrito en forma única como.

$$\alpha = a_1\alpha_1 + a_2\alpha_2 + ... + a_{m-1}\alpha_{m-a}$$

Donde $\alpha_i \in \{0, 1\}$. Al conjunto $\{\alpha_1, ..., \alpha_{m-1}\}$ se le denomina base de GF(2^m).

Bases polinomiales

Una base polinomial es una representación del campo GF(2^m) que esta compuesto por un conjunto de polinomios irreducibles sobre GF(2) con grado menor a m, de tal forma que cualquier elemento del campo es combinación lineal de polinomios irreducibles $f(x)$

$$\{f_m(x), f_{m-1}(x), ..., f_1(x), f_0(x)\}$$

Existe una base canónica de GF(2^m) y es el conjunto formado por los polinomios.

$$\{x^{m-1}, x^{m-2}, ..., x^2, x, 1\}$$

Como GF(2^m) esta compuesto por todos los polinomios sobre $\mathbb{Z}_2$ con grado menor a m, GF(2^m) queda expresado en términos de su base canónica de la siguiente manera.

$$\text{GF}(2^m)=\{f(x) = a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a_0 | a_i \in \{0, 1\}\}$$

La base canónica permite representar los elementos de GF(2^m) como vectores de coeficientes de tamaño m de $f(x)$, de modo que.

$$\text{GF}(2^m)=\{(a_{m-1}, ..., a_1, a_0) | a_i \in \{0, 1\}\}$$

Aritmetica en campos de Galois binarios

DEFINICIÓN 9 (OPERACIONES EN GF(2^m))
Sean $a(x)$=($a_{m-1}, ..., a_1, a_0$), $b(x)$=($b_{m-1}, ..., b_1, b_0$)$\in$ GF(2^m) y $f(x)$ polinomio irreducible sobre GF(2^m), se definen las siguientes operaciones en GF(2^m): [6]

- **Suma en GF(2^m):** ($a_{m-1}, ..., a_1, a_0$)+($b_{m-1}, ..., b_1, b_0$)=($a_{m-1} \oplus b_{m-1}, ..., a_1 \oplus b_1, a_0 \oplus b_0$).
- **Producto en GF(2^m):** ($a_{m-1}, ..., a_1, a_0$)•($b_{m-1}, ..., b_1, b_0$)=$a(x) \bullet b(x)$ mod $f(x)$.

A continuación se muestra el pseudocódigo correspondiente a la operación de suma en GF(2^m). Siendo una operación secuencial se requieren m operaciones XOR, por lo que su orden de complejidad es O(m). [7]

Algoritmo 1 Algoritmo de la suma en GF(2^m)

Entrada: $f(x) = \sum_{i=0}^{m-1} a_i x^i$, $g(x) = \sum_{i=0}^{m-1} b_i x^i \in$ GF(2^m).
Salida: $h(x) = \sum_{i=0}^{m-1} c_i x^i \in GF(2^m)$.
1: **para** $i = 0$ hasta m **hacer**
2: $c_i \rightarrow (a_i + b_i)$ mod 2.
3: **fin para**

La multiplicación de dos elementos en el campo finito GF(2^m) es una operación más compleja y requiere de dos pasos más para su cálculo un producto y una reducción.

En la multiplicación de polinomios en GF(2^m), es posible que el resultado contenga elementos que se encuentren fuera del campo es decir que contengan términos con grados mayor o igual a m, por que es necesario aplicar la operación de reducción mediante un polinomio $m(x)$ necesariamente irreducible y de grado m. [7]

Consideremos dos elementos de GF(2^m), digamos $f(x) = \sum_{i=0}^{m} a_i x^i$ y $g(x) = \sum_{i=0}^{m} b_i x^i$ y sea $m(x)$ polinomio irreducible, entonces.

$$\begin{aligned} f(x) \bullet g(x) &= f(x)g(x) \text{ mod } m(x) \\ &= f(x) \textstyle\sum_{i=0}^{m} b_i x^i \text{ mod } m(x) \\ &= \textstyle\sum_{i=0}^{m} b_i (x^i f(x) \text{ mod } m(x)) \\ f(x) \bullet g(x) &= b_0 f(x) + b_1 x f(x) \text{ mod } m(x) + ... + b_{m-1} x^{m-1} f(x) \text{ mod } m(x) \end{aligned}$$

Notemos que en cada termino de la suma es necesario:

- Una multiplicación de un elemento de GF(2) por un elemento de GF(2^m) consume m multiplicaciones.
- Una suma en GF(2^m) requiere consume m sumas.
- Una multiplicación por x.
- 1 operación de suma.

A continuación se muestra el pseudocódigo correspondiente a la operación de multiplicación en GF(2^m). Siendo una operación secuencial se requieren 2m-1 operaciones XOR y 2m operaciones AND. [7]

Algoritmo 2 Algoritmo de la multiplicación en GF(2^m)

Entrada: $f(x) = \sum_{i=0}^{m-1} a_i x^i$, $g(x) = \sum_{i=0}^{m-1} b_i x^i \in$ GF(2^m).
Salida: $h(x) = \sum_{i=0}^{m-1} c_i x^i \in GF(2^m)$.
1: $c \leftarrow 0$
2: **para** $i = 0$ hasta $m - 1$ **hacer**
3: $c \leftarrow b_i f(x) + c$
4: $f(x) \leftarrow f(x)(x^i)$ mod $P(x)$
5: **fin para**

El campo finito GF(2^8)

DEFINICIÓN 10 (CAMPO GF(2^8))
GF(256) ó GF(2^8) es un campo de Galois con característica 2 y grado de extensión 8, el cual consta 256 polinomios $p(x)$ con grado menor a 8 de la forma: [46]

$$p(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_01$$
correspondiente al byte $(b_7, b_6, b_5, b_4b_3, b_2, b_1, b_0)$

El campo finito GF(2^8) es de especial importancia en el diseño de algoritmos criptográficos ya que es el campo ideal para trabajar elementos que pueden representar unidades de almacenamiento de una computadora (bytes). Un XOR entre dos bytes corresponde a una suma mientras que la multiplicación es una operación complicada sin embargo una de las ventajas de GF(2^8) es que permite expresar multiplicaciones entre dos bytes en términos de operaciones propias del álgebra booleana. Como el campo GF(2^8) es el conjunto cociente $\mathbb{Z}_2$[x]/$_{m(x)}$ de clases residuales módulo $m(x)$ es posible construirlo si se cuenta con un elemento generador y $m(x)$ cualquier polinomio irreducible, por convención siempre se toma el primer polinomio irreducible de grado igual a 8 para construir el campo mientras que el elemento generador puede variar, dicho elemento corresponde al polinomio $m(x) = x^8 + x^4 + x^3 + 1 = (100011001) = (11B)_{hex}$. La idea de desarrollar estas construcciones es la de poder diseñar un algoritmo de multiplicación simple en GF(2^8), en el sentido que quede en términos de compuertas y desplazamientos. [6]

Sean $f(x)$ y $g(x) \in$ GF(2^8), entonces.

$$\begin{aligned} f(x) \bullet g(x) &= \textstyle\sum_{i=0}^{7} b_i(x^i f(x) \text{ mod } m(x)) \\ &= \textstyle\sum_{i=0}^{7} b_i(x^i \sum_{j=0}^{7} a_j x^j \text{ mod } m(x)) \\ &= \textstyle\sum_{i=0}^{7} (\sum_{j=0}^{7} b_i a_j x^{j+i} \text{ mod } m(x)) \\ f(x) \bullet g(x)(x) &= \textstyle\sum_{i=0}^{7} (\sum_{j=0}^{7} b_i a_j x^{j+i} \text{ mod } x^8 + x^4 + x^3 + 1) \end{aligned}$$

En la j-ésima suma parcial, como $\sum_{j=0}^{7} b_i a_j x^{j+i} = x^{i-1} x \sum_{j=0}^{7} x^j b_i a_j$ entonces se toma como caso base el producto $x \sum_{j=0}^{7} b_i a_j x^j = \sum_{j=0}^{7} b_i a_j x^{j+1}$ por lo tanto $xf(x)$ equivale a sumar un grado a cada termino de $f(x)$. Ya que $f(x)$ es un byte el producto de $f(x)$ con x se traduce a un desplazamiento aritmético a la izquierda de $f(x)$, pero pueden ocurrir dos casos en un producto de este estilo.

Caso 1 El grad($xf(x)$) $<$ 8 entonces $xf(x) = f(x) \ll 1$ con lo que la suma parcial queda dentro de GF(2^8).

Caso 2 El grad($xf(x)$) $\geq$ 8 entonces se hace ($f(x) \ll$1) y se le aplica modulo $m(x)$, desarrollando tenemos:

$$xf(x) = a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x$$

Desarrollando el cociente $xf(x)$ mod $x^8 + x^4 + x^3 + 1$

$$\begin{array}{r|l} & a_7 \\ \hline x^8+x^4+x^3+1 & a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \\ \hline & -a_7x^8 - a_7x^4 - a_7x^3 - a_7 \text{ (como } -a = +a \text{ mod 2 se intercambian los signos)} \\ & +a_7x^8 + a_7x^4 + a_7x^3 + a_7 \\ \hline & (a_6\oplus 0)x^7+(a_5\oplus 0)x^6+(a_4\oplus 0)x^5+(a_3\oplus a_7)x^4+(a_2\oplus a_7)x^3+(a_1\oplus 0)x^2+(a_0\oplus a_7)x+(a_7\oplus 0) \end{array}$$

Observemos que para que $xf(x)$ quede fuera del campo necesariamente a_7 es 1, con lo que resulta:

$xf(x)$ mod $m(x)$=$(a_6\oplus 0)x^7+(a_5\oplus 0)x^6+(a_4\oplus 0)x^5+(a_3\oplus 1)x^4+(a_2\oplus 1)x^3+(a_1\oplus 0)x^2+(a_0\oplus 1)x+(a_7\oplus 0)$.

$xf(x)$ mod $m(x)$=$(a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x + 0) + (m(x) + b_7x^8)$

Pero $m(x) = (100011001)$ entonces $xf(x)$ mod $m(x)$=$(f(x) \ll$1$)\oplus(1B)_{hex}$.

Así obtenemos el cálculo de una suma parcial trivial para el cálculo de una suma parcial con un monomio de grado mayor a 1 solo es necesario aplicar la operación anterior tantas veces como el grado del monomio, notemos que el cálculo de una suma parcial $\sum_{j=0}^{7} b_i a_j x^{j+i}$ equivale a calcular la suma parcial anterior más el caso base de modo que si se acumula el resultado de la suma parcial a una suma total si el coeficiente b_i correspondiente a g es uno, si no se deberá calcular la siguiente suma parcial y guardar el resultado de la suma parcial actual. con lo que queda bien definido el algoritmo de la multiplicación en GF(2^8).

Observación Siendo operaciones secuenciales en el mejor caso se requieren cero operaciones, mientras que en el peor caso se requieren 16 operaciones XOR (8 por cada suma parcial y 8 por cada reducción modular), 16 Corrimientos y 16 compuertas AND, por lo que su orden de complejidad es $O(m)$ (lineal) [47].

Algoritmo 3 Algoritmo de la multiplicación en GF(2^8)

Entrada: Byte A, Byte $B \in$ GF(2^m).
Salida: Byte $R \in$ GF(2^m).

```
1: Byte R ← (00)_hex
2: Byte T ← (00)_hex
3: mientras A ≠ (00)_hex hacer
4:    si (A and 01_hex) ≠ (00)_hex entonces
5:       R ← R ⊕ B
6:    fin si
7:    T ← B And (08)_hex
8:    B ← B ≪ 1
9:    si T ≠ 00_hex entonces
10:      B ← B ⊕ (1B)_hex
11:   fin si
12:   A ← A ≫ 1
13: fin mientras
14: devolver R
```

2.5. Aplicación de los campos finitos a la criptografía

En criptografía, los campos de Galois juegan un papel muy importante en la criptografía moderna ya que en principio la idea de diseñar algoritmos criptograficos basados en campos finitos es la de extender las propiedades de los cifradores afines a elementos de campos de Galois. Entre los sistemas de cifrado más representativos que basan su funciones de cifrado en campos de Galois actualmente se encuentran el Data Encryption Standard (DES), Advanced Encryption Standar (AES), Twofis, Safer y Curvas elípticas.

Algoritmo DES

Este algoritmo maneja llaves de 64 bits es decir elementos de GF(2^6). de los que 8 bits (un byte) se utilizan como control de paridad (para la verificación de la integridad de la clave). Cada uno de los bits de la clave de paridad (1 cada 8 bits) se utiliza para controlar uno de los bytes de la clave por paridad impar, es decir, que cada uno de los bits de paridad se ajusta para que tenga un número impar de ”1”dentro del byte al que pertenece. Por lo tanto, la clave tiene una longitud ”útil”de 56 bits, es decir, realmente sólo se utilizan 56 bits en el algoritmo. [8]

El algoritmo se encarga de realizar combinaciones, sustituciones y permutaciones entre el texto a cifrar y la clave, asegurándose al mismo tiempo de que las operaciones puedan realizarse en ambas direcciones (para el descifrado).

La clave es codificada en 64 bits y se compone de 16 bloques de 4 bits, generalmente anotadas de k_1 a k_16. Dado que ”solamente”56 bits sirven para el cifrado, puede haber hasta 2^56 claves útiles.

Durante la construcción de este algoritmo, se consideraron permutaciones inducidas por un polinomio $f(z) \in$ GF(2)[z] irreducible, con el cual se construye el campo de $GF(2)[z]_{/f(z)}$ donde $f(z) = z^8 + z^4 + z^3 + z^2 + 1$.

Una de las partes medulares de los sistemas de cifrado simétricos son las llamadas **Cajas de sustitución (S-Boxes)**. Estas Cajas son básicamente funciones sobre campos de Galois. Por ejemplo en el sistema DES, una caja de sustitución es una función.

$$\text{S-Box:} GF(2^2) \text{x} GF(2^2) \rightarrow GF(2^4)$$

El algoritmo esta diseñado para cifrar y descifrar bloques de datos de 64 bits bajo una llave de 64 bits. El descifrado debe llevarse acabo por la misma llave con la cual se llevo acabo el cifrado. Un bloque que se somete al cifrado es sujeto a una permutación inicial IP, después se aplica un cálculo dependiente de la clave durante 16 ronda y finalmente se le aplica una permutación final PF^{-1}. El cálculo dependiente de la llave puede ser definido en términos de una función f, llamada función de cifrado y adicional mente es necesario añadir una función KS (Key Schendule ó Calendarización de Llave) que somete la clave de cifrado a una transformación de sus bits por cada ronda de cifrado.

Algoritmo AES

Son varias las estructuras algebraicas usadas en el diseño del sistema de cifrado AES, en las cuales los campos de Galois juegan un papel muy importante. La principal y básica estructura algebraica usada en el sistema AES es el campo binario de Galois GF(2^8). Por consiguiente, a los bytes se les puede representar como polinomios en una indeterminada de grado a los más 7 con coeficientes binarios existe otra forma de representar a los bytes usando campos de Galois ya que es muy importante en el cifrado AES ya que sobre esta se define las operaciones de cifrado. Esta estructura algebraica es la del campo finito. [8]

$$G(2^8) = GF_2[x]_{/f(x)} \text{ con } f(x) = x^8 + x^4 + x^3 + x + 1$$

El modo de operación del sistema AES es por medio de estados. Un estado se representa como un arreglo rectangular 4xN donde cada entrada es un byte y N depende de la longitud del bloque a cifrar, por ejemplo si N = 4 se tiene un arreglo de 16 bytes. Así, un estado se puede representar como un elemento del espacio vectorial $[GF(2^8)]^{4N}$. El campo de Galois GF(2^8) también se usa para definir la caja de sustitución y las matrices de cifrado/descifrado del sistema AES. Otra propiedad importante de este campo de Galois que se usa en el sistema AES es que el grupo GF(2^8)=GF(2^8) $\{0\}$ es cíclico de orden $2^8 - 1 = 255$. Por consiguiente un byte distinto de cero se puede identificar como una potencia de un generador de este grupo cíclico. [8]

$$\phi{:}GF(2^8) \to GF(2^8)$$

la cual es una permutación. Dado que el grupo multiplicativo de este campo tiene orden 255, esta permutación se puede expresar como $\phi(x) = x^{254}$, es decir, $\phi(x)$ es un polinomio de permutación.

El modo de operación del sistema AES es por medio de estados. Un estado se representa como un arreglo rectangular 4xN donde cada entrada es un byte y N depende de la longitud del bloque a cifrar, por ejemplo si N = 4 se tiene un arreglo de 16 bytes. Así, un estado se puede representar como un elemento del espacio vectorial [GF(2^8)]^{4N} [8]

El algoritmo Twofish

Este es otro sistema de cifrado de clave privada donde los campos de Galois se ponen de manifiesto. Este sistema maneja bloques de texto y llaves formados por bytes, es decir, elementos del espacio vectorial GF(2^8), los cuales se concatenan para formar palabras, vistas como elementos de GF(2)^{32}. Otra componente de Twofish es una matriz de tipo MDS (Maximum Distance Separable) cuyas entradas son bytes. [8]

Para realizar operaciones es necesario identificar a los bytes con elementos de un campo. Este campo es GF(2^8) = GF(2)[x]$_{/v(x)}$, donde $v(x) = x^8 + x^4 + x^3 + 1 \in$ GF(2)[x] es irreducible sobre GF(2). Este sistema además usa una matriz de tipo Reed-Solomon cuyas entradas son también bytes. Para efectuar las operaciones se identifica el espacio GF(2^8) con el campo de Galois con 28 elementos, pero ahora descrito de la siguiente manera: GF(2^8)=GF(2)[x]$_{/w(x)}$ donde $w(x) = x^8 + x^4 + x^3 + 1$.

El algoritmo Safer

En sistema de cifrado SAFER de clave privada (Secure And Fast Encryption Routine) también usa fuertemente los campos de Galois, particularmente el campo GF(257). El elemento 45 $\in$ GF(257) es un generador del grupo cíclico GF(257)* y este grupo se puede identificar con $\mathbb{Z}_{256}$, con lo cual se define la permutación. [8]

$$exp : \mathbb{Z}_{256} \to \mathbb{Z}_{256}, exp(x) = 45^x$$

que juega un papel importante en este sistema de cifrado. La permutación inversa de exp se denota por log, que corresponde justamente a la función de descifrado.

El algoritmo basado en Curvas Elípticas

Las curvas elípticas definidas sobre un campo algebraicamente cerrado han sido objeto de estudio desde hace mucho tiempo por diversos motivos. Recientemente N. Koblitz y V. Miller, en forma independiente, presentaron un sistema de cifrado de llave publica basado en el grupo de puntos (racionales) de una curva elíptica definida sobre un campo de Galois, el cual, para aplicaciones, se toma el campo de la forma GF(2^m) para diversos valores de m, por ejemplo m=160. [8]

En este sistema de cifrado los campos de Galois se manifiestan de varias maneras. Para definir una curva elíptica se necesita un campo, el cual para propósitos de cifrado debe ser de Galois, GF(p^m), elementos, p un número primo y n $>$ 0 un entero. La relación que define una curva elíptica en general es un polinomio cubico en dos indeterminadas con coeficientes en un campo. Si el campo sobre el cual se esta trabajando es de Galois, GF(q), el polinomio que define a la curva elíptica se puede reducir a una relación de la siguiente forma (dependiendo de la característica del campo):

$$f(x, y) = y^3 - ax^3 - bx - v \in GF(q)[x, y]$$

y la curva elíptica, E, sobre GF(q) se define como:

$$E = \{P = (x, y) \in GF(q)\text{x}GF(q) | f(x, y) = 0\}$$

En el conjunto E (o mejor dicho, de los GF(q)-puntos racionales de E, se puede definir una operación que le da una estructura de grupo abeliano a este conjunto. Si P;Q 2 E cuyas coordenadas son elementos de GF(q), las coordenadas de P+Q son en general funciones racionales en las coordenadas de P y Q, las cuales a su vez son elementos del campo de Galois GF(q). En este contexto una de las cuestiones importantes de los campos de Galois es realizar aritmética rápida para obtener las coordenadas explicitas del elemento P+Q, en particular nP, donde n $>$ 0 es un entero.

La primera aparición de la criptografía con curvas elípticas fue en el estándar X9.62, con la adopción del esquema de firma digital ECDSA (Elliptic Curve Digital Signature Algorithm) en enero de 1999, Algunas de las características más iniciales que se adoptaron fueron una longitud mínima para claves de 80 bits y el uso de bases polinomiales sobre GF(2^m) [9].

Autómatas celulares

3. Autómatas celulares

3.1. Autómatas celulares

Un autómata celular, es un sistema dinámico discreto que evoluciona en iteraciones a través de una regla determinística, tal como un sistema dinámico; las variables del sistema cambian como una función de sus valores actuales. Pueden ser vistos como un proceso de cálculo en paralelo, donde los datos son la configuración inicial. Otra aproximación es que un autómata celular es un "universo lógico con su propia física local". Tal como un universo de autómata celular; no obstante su construcción matemática, son capaces de soportar comportamientos complejos. El concepto de AC lleva implícitamente asociado otros conceptos, como espacio y localidad de influencia. Se asume que el sistema representado está distribuido en el espacio y que regiones cercanas tienen mayor influencia entre si, que otras que se encuentren apartadas dentro del sistema [10].

Los autómatas involucran cuatro elementos en su construcción [11]:

1. **El espacio celular**. Los autómatas celulares están definidos por un arreglo de células de dimensión d, llamado espacio celular; donde cada uno de los elementos del arreglo se denomina célula, que es un término tomado de las ciencias biológicas como algunos otros tales como "evolución", "vida", "muerte"; que son adecuados para comprender el comportamiento individual de los elementos. El espacio celular, en principio y tal como fue definido originalmente, es infinito en todas sus direcciones, y posteriormente debido principalmente a las limitaciones prácticas, el espacio celular puede tomar diversas configuraciones.

2. **Los estados de las células**. El estado que cada célula tendrá en el siguiente paso del tiempo está determinado por el estado actual de ella misma y de las células vecinas más cercanas en un radio de vecindad predeterminado. El número de estados para un autómata celular lineal puede ser variable, se representa por medio de la variable k, es decir, la cardinalidad del conjunto K es igual a k, lo cual se escribe como $\#(K) = k$.

3. **La configuración de vecindades**. La manera en que las células actúan dentro del espacio celular, está determinada por como son afectadas por el entorno que las rodea. A este entorno se le llama vecindad. Así, la vecindad de una célula, llamada célula en transición, o célula central x_c , es el conjunto de células cercanas a ella a una distancia $d(x_c, x_i) \leq r$, además de la propia célula central, a r se le conoce como el radio de la vecindad.

4. **Función de transición local**. Cada célula del espacio celular, toma su valor en dependencia de la configuración de su vecindad. Si K es el conjunto de estados.

Formalmente un autómata celular se define como

DEFINICIÓN 11 (AUTÓMATA CELULAR)
Un autómata celular es una tupla $< L, K, V, \delta(x_c, V) >$ [12].

1. **L**: el Lattice ó Espacio Celular es una retícula regular L tal que si $d \in \mathbb{N}$ entonces $L = \{c|\mathbb{Z}^d\}$, para un lattice L de dimensión d.

2. **K**: conjunto de estados, es el conjunto finito de valores que pueden tomar las células c tal que $c \in L$.

3. **V**: vecindad, conjunto finito de células que definen la vecindad para una célula, es decir es el conjunto de celular para las cuales la célula central x_c es el punto de referencia para el área de influencia de radio r. $V = \{x_i|d(x_c, x_i) \leq r\} \cup \{x_c\}$

4. $\delta(x_c, V)$: $K^d \rightarrow K$. Función en términos de la célula central x_c y las células x_i tal que $x_i \in V$, que es aplicada simultáneamente a todas las células.

Condiciones de frontera

La definición de lattice por si misma nos permite considerar lattices de tamaño infinito, pero en la práctica esta implementación resulta imposible, es por eso que los AC son representados como sistemas en espacios finitos, a estas condiciones que nos permiten limitar el espacio de operación del AC las llamamos condiciones de frontera. Los tipos de condiciones de frontera que se pueden manejar son cuatro [12]:

- **Frontera periódica**. Condición de frontera que permite tomar el espacio que utilizamos para representar el AC de manera continua, uniendo los extremos, la vecindad incluye a la célula i, sin embargo, la actualización por la función no depende del estado de i en el tiempo t, por lo general se busca que esta vecindad respete la simetría respecto a la célula central, aunque no es obligatorio.

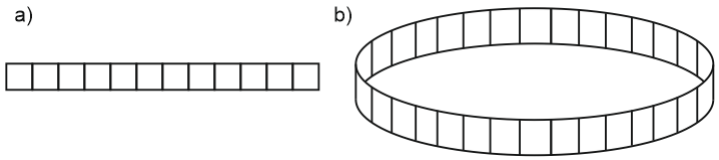

Figura 2: Representación del lattice de un AC-unidimensional, a) frontera infinita, b) frontera periódica [12].

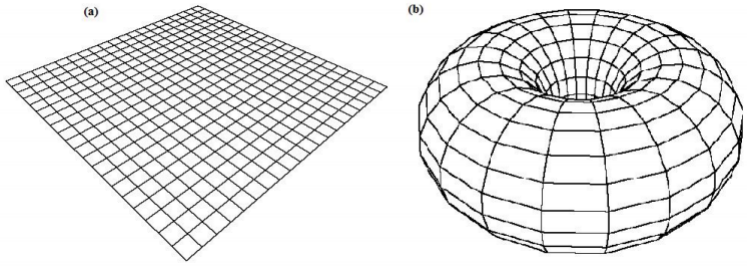

Figura 3: Representación del lattice de un AC-bidimensional, a) frontera infinita, b) frontera periódica [12].

- **Frontera fija**. Condición de frontera que completa la vecindad con células virtuales con un valor asignado.
- **Frontera adiabática**. Condición de frontera obtenida por la duplicación del valor de la célula cercana a la célula virtual.
- **Frontera reflectante**. Condición de frontera obtenida de copiar el valor de otros vecinos en la célula virtual.

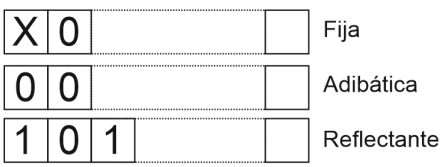

Figura 4: Representación del lattice de un AC con frontera fija, adibática y reflectante [12]

.

Funciones de transición totalísticas

DEFINICIÓN 12 (FUNCIÓN DE TRANSICIÓN TOTALÍSTICA)
El termino se refiere a aquella clase de funciones de transición que son función de la suma de los estados de las celdas en la vecindad. En las funciones de transición totalísticas se suman los valores de los elementos que forman la vecindad y todas aquellas vecindades que correspondan a esa suma, evolucionan al mismo valor. Una función de transición totalística es de la siguiente forma [48], [49].

$$C_i(t+1) = \delta(\sum_{j \in V} C_j(t))$$

DEFINICIÓN 13 (FUNCIÓN DE TRANSICIÓN SEMI-TOTALÍSTICA)
Por su parte las funciones de transición semi totalísticas son aquellas en las cuales el estado del sitio central depende separadamente en la suma de los estados de las celdas de la vecindad y en el estado del propio sitio. En la reglas semi-totalísticas igualmente se realiza una suma, pero ésta solo es realizada con los vecinos sin tomar en cuenta la célula central. La célula central es considerada solo para determinar en que condición debe evaluarse la suma [48], [49].

$$C_i(t+1) = \delta(\sum_{j \in V} C_i(t), C_j(t))$$

3.1.1. Autómatas celulares unidimensionales

Un autómata celular unidimensional consta de un arreglo lineal finito de celdas o células (ver figura 3.6), una de las características por las cuales a los autómatas celulares en una dimensión también se les conoce como Autómatas Celulares Lineales. 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.) Puesto que la naturaleza de los estados no es relevante. Para el procesamiento interno de las evoluciones del autómata celular lineal a través del tiempo lo más conveniente es utilizar números, esto por cuestiones computacionales, y para la representación final de esas evoluciones es útil utilizar diferentes colores, esto por cuestiones de visualización. Aunque la naturaleza de los estados no influye en el comportamiento del autómata celular lineal, la interacción que mantienen las células durante la evolución por medio de esos estados sí influye de manera directa en el comportamiento del autómata celular lineal [13].

DEFINICIÓN 14 (AUTÓMATA CELULAR UNIDIMENSIONAL)
Un autómata celular unidimensional es un autómata celular $AC_1 =< L, K, V, \delta(x_c, V) >$ tal que $dim(AC(L)) = 1$. La figura 2 ilustra claramente el lattice de un AC de dimensión 1.

En un autómata celular unidimensional el tamaño total de una vecindad es igual a $2r + 1$ incluyendo a la célula central.

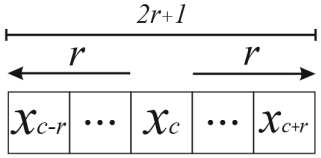

Figura 5: Representación de una vecindad en un AC de dimensión 1 de radio r

Sea $AC_1 =< L, K, V, \delta(x_c, V) >$ autómata celular unidimensional, $F = \{\delta | \delta$ es función de transición de $AC_1\}$ (F es el conjunto posibles de funciones de transición de AC_1) y $V_r = \{$v|v es

vecindad} (el conjunto de configuraciones de vecindades posibles con radio r), entonces para todo AC_1 si $\#(K) = k$, se cumple que el número posibles de configuraciones de vecindades es $\#(V_r) = K^{2r+1}$ y por lo tanto $\#(F) = k^{k^{2r+1}}$.

DEFINICIÓN 15 (TRANSICIÓN DE UN AUTÓMATA CELULAR UNIDIMENSIONAL)
Sea AC_1=$< L, K, V, \delta(x_c, V) >$ un autómata celular unidimensional, entonces $\delta(x_c, V)$

$\delta : K^{2r+1} \to K$ tal que $x(t+1)_c = \delta(x(t)_{c-r}, ..., x(t)_{c-1}, x(t)_c, x(t)_{c+1}, ..., x(t)_{c+r}), \forall x_c \in L$

Es decir que la función de transición toma una configuración de vecindad con centro en la célula x_c en el instante t, y le asigna un valor de K a la célula x_c en el instante t+1.

Como $\#(F)$ es un número bastante grande, es necesario crear un forma sistemática de analizar las funciones de transición, Wolfram propone la siguiente nomenclatura para AC's de una dimensión.

DEFINICIÓN 16 (NOMENCLATURA DE WOLFRAM)
Sea AC_1=$< L, K, V, \delta(x_c, V) >$ un autómata celular unidimensional y $F = \{\delta | \delta : K^{2r+1} \to K\}$ entonces el par AC_1=(R, r) determina de manera univoca a un AC de dim 1 donde r es el radio de la vecindad y R es una indice que depende de δ ya que $\forall\ \delta \in F$ le podemos asignar un único número natural R de base k tal que $R \in I_{(\#(F)-1)} = \{0, ..., k^{k^{2r+1}} - 1\}$ siguiendo la siguiente función biyectiva [14].

$R(\delta) : F \to \mathbb{N}_{(k)}$ con regla de correspondencia $R(\delta)$=$\sum_{n=0}^{k^{2r+1}-1}(x_n)(k^n)$.

Donde x_n es imagen de δ y $0 \leq R(\delta) \leq k^{k^{2r+1}} - 1$.

De esta manera es posible identificar de manera única aun autómata celular de dimensión 1 denotado con el par AC_1(R,r), por medio su función de transición ya que esta tiene asignado un único número natural R de base k entre 0 y $\#(F)$. Cabe mencionar que si bien este esquema de notación es único y conciso es puramente formal, es decir no provee indicación alguna sobre las propiedades de la funciones de transición resultante de las reglas correspondientes. El asociar alguna propiedad dinámica sutil con R no es en general tarea fácil [49].

Ahora presentaremos un elemento de un AC que a pesar de que no formar parte de la definición es igual de importante, ya que dicho influye a largo plazo en comportamiento de un autómata y es la configuración inicial, de hecho distintas instancias de configuraciones iniciales pueden producir distintos comportamientos dependiendo de la regla. Una configuración inicial es un vector de valores de valores de K en L no nulo, en el instante 0.

DEFINICIÓN 17 (CONFIGURACIÓN INICIAL)
Sea a un AC_1=$< L, K, V, \delta(x_c, V) >$ de dimensión 1, la configuración inicial C_1 de un AC es un parámetro de inicialización tal que.

$C_i \in K^+$ tal que $C_i \subset L$ en t=0.

Autómatas elementales

Los autómatas celulares elementales son una clase más simple de los autómatas celulares unidimensionales con radio entero (existe una clase aun más simple de radio fraccionario que se explica mas adelante).

DEFINICIÓN 18 (AUTÓMATA CELULAR ELEMENTAL)
Sea AC_1=$< L, K, V, \delta(x_c, V) >$ un autómata celular, se dice que AC_1 es elemental si se cumple que:

- $dim(L) = 1$, es decir que AC_1 es unidimensional.
- $K = \{0, 1\}$, es decir las células solo pueden tomar valores 0 ó 1. Es claro que $\#(K) = k = 2$.
- $V = x_{c-1}, x_c, x_{c+1} \forall x_c \in L$, es decir que el radio de la vecindad es 1.

Los autómatas celulares elementales tienen dos valores posibles para cada celda (0 o 1), y las reglas que dependen sólo de los valores de vecinos más cercanos. Como resultado, la evolución de un autómata celular elemental completamente puede ser descrita por una tabla que especifica el estado de una célula dada tendrá en la próxima generación basado en el valor de la célula a su izquierda, el valor de la propia célula, y el valor de la célula a su derecha. Siguiendo las definiciones anteriores un AC elemental cumple lo siguiente [50]:

1. Como $k = 2$ y el radio $r = 1$ entonces el conjunto V_1 de posibles configuraciones de vecindad de radio 2 tiene cardinalidad $\#(V_1) = 2^{2+1} = 2^3 = 8$, donde V_1 son todas las posibles permutaciones de tres elementos donde cada elemento puede valer 0(blanco) ó 1(negro), V_1 por extensión es:

Figura 6: Conjunto V_1 de configuraciones de vecindades posibles en un AC elemental

2. Como $\#(V_1) = 8$ entonces $\#(V_1) = 2^8 = 256$, por lo tanto para cada función de transición δ en un AC elemental se cumple que le puede asignar un número base 2 entre 0 y 255.

$$R(\delta) = \sum_{i=0}^{7}(\delta(V_i))(2^i), \forall\, Vi \in V, 0 \leq R(\delta) \leq 255.$$

Consideremos la siguiente función de transición: $\delta : V_1 \rightarrow K$ donde la regla de correspondencia se muestra en la imagen 7 que ilustra los valores posibles de las tres celdas vecinas se muestran en la fila superior de cada panel, y el valor resultante de la célula central lleva en la próxima generación.

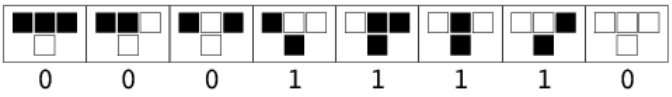

Figura 7: Función de transición de un AC. Figura tomada de [50].

Aplicando la notación de Wolfram tenemos que $R(\delta) = \delta(V_{1,1})2^0 + \delta(V_{1,2})2^1 + \delta(V_{1,3})2^2 + \delta(V_{1,4})2^3 + \delta(V_{1,5})2^4 + \delta(V_{1,6})2^5 + \delta(V_{1,7})2^6 + \delta(V_{1,8})2^7 = 0 * 1 + 1 * 2 + 1 * 4 + 1 * 8 + 1 * 16 + 0 * 32 + 0 * 64 + 0 * 128$ por lo tanto $R(\delta) = 30$, Entonces el AC elemental de que hablamos en la figura 7 es aquel cuya regla de correspondencia esta inequívocamente asociada al número $30_{(2)}$ y el AC se denota como $AC(30_{(2)}, 1)$. Por simplicidad usualmente se les denomina a los autómatas por su número de su regla, en este caso el AC seria la **regla 30**.
A continuación se presentan ejemplos de algunas otras reglas para un AC elemental.

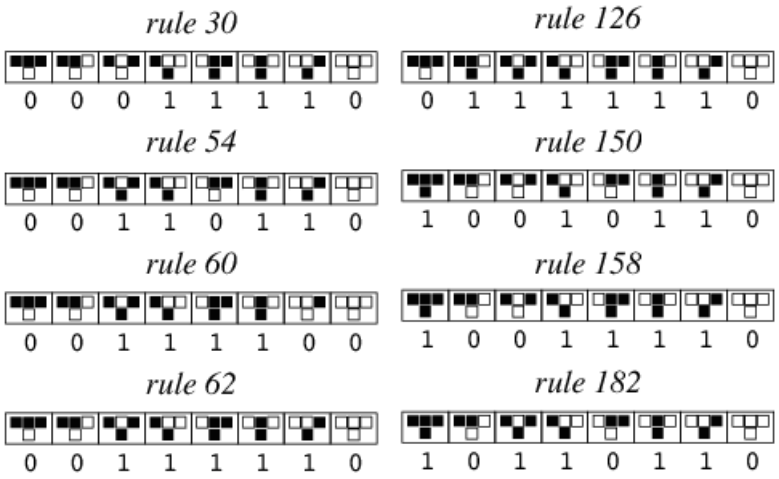

Figura 8: Ejemplos de funciones de transición para un AC unidimensional. Imagen tomada de [50]

Dadas las reglas anteriores la siguiente imagen se muestra la evolución del sistema al cabo de 15 transiciones cuando la configuración inicial es 1 con cada una de las reglas que se muestran en la imagen 8.

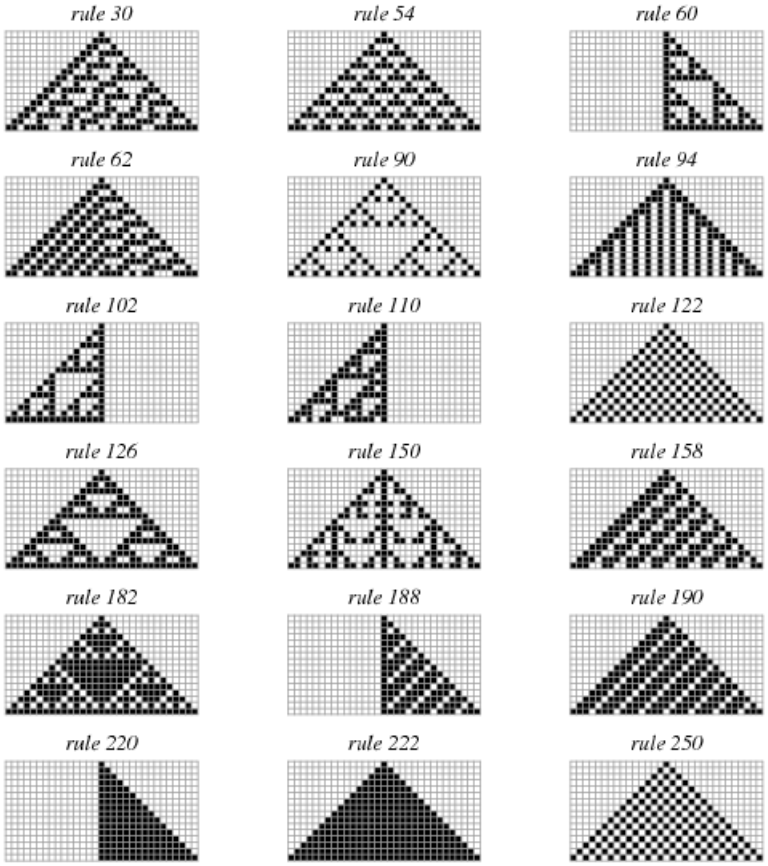

Figura 9: Evolución para cada una de la reglas anteriores. Imagen tomada de [50]

3.1.2. La Clasificación de Wolfram

Como se puede ver en la imagen 13, las gráficas anteriores muestran reglas de autómatas exhiben comportamientos muy distintos unos de otros ya que generan diferentes patrones al

cabo de 15 transiciones, dicho comportamiento es una característica cualitativa cada regla. Unos de los primeros trabajos realizados que buscaban dar una clasificación de reglas fueron los estudios realizados por Stephen Wolfram, AC unidimensionales con dos o tres estados y el análisis morfológico independiente de cualquier configuración inicial de las gráficas de las evoluciones a lo largo del tiempo, Wolfram clasificó los autómatas en 4 clases, dependiendo del nivel de complejidad.

- Clase I La evolución conlleva a un estado uniforme. Después de transcurrido un cierto número de generaciones, todas las células del autómata convergen a un solo estado.

Figura 10: Regla 234

- Clase II La evolución lleva a un conjunto de estructuras simples que son estables o periódicas.

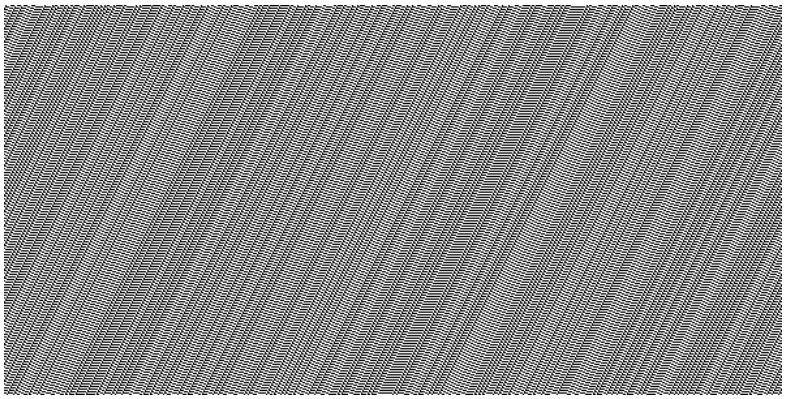

Figura 11: Regla 53

- Clase III La evolución lleva a un patrón caótico.

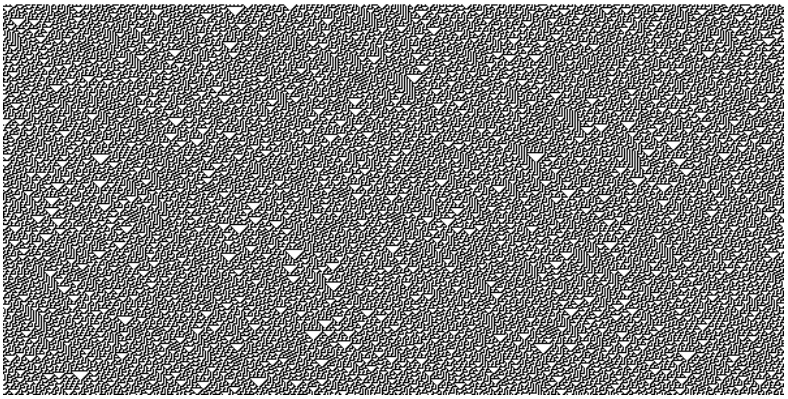

Figura 12: Regla 30

- Clase IV La evolución lleva a estructuras aisladas que muestran un comportamiento complejo (ni completamente caótico, ni completamente ordenado, sino en la línea entre uno y otro, este suele ser el tipo de comportamiento más interesante que un sistema dinámico puede presentar).

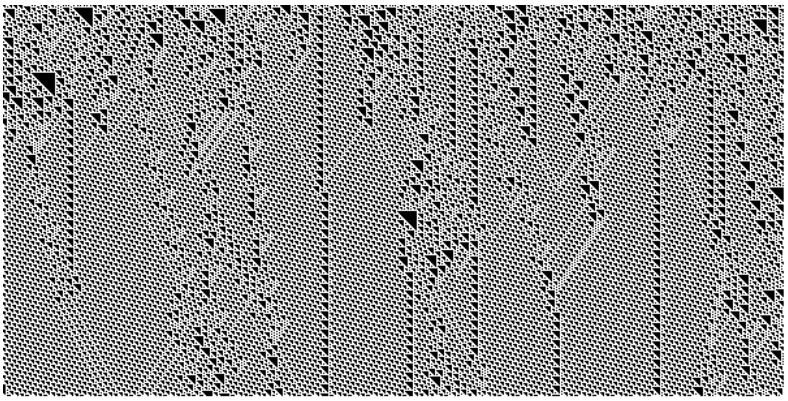

Figura 13: Regla 193

Automatas con radio 1/2

A pesar de que en un principio la definición de autómata celular no admite vecindades de radio fraccionario, es posible modificar el lattice realizando un corrimiento en cada evolución de modo que la vecindad de influencia se ajuste a un radio de un medio con lo que se pierde el sentido de célula central, un AC que consta de radio 1/2 induce un lattice como se muestra en la siguiente figura.

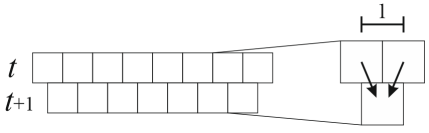

Figura 14: Lattice con localidad de influencia 1/2

Al igual que en los autómatas celulares unidimensionales, los AC's también lo hay elementales, es decir con K=$\{0, 1\}$. Un AC elemental de radio 1/2 es la clase mas simple de autómata celular unidimensional y cumple lo siguiente:

- El conjunto de configuraciones de vecindades posibles es $\#(V_{1/2}) = 2^2 = 4$.

Figura 15: El conjunto de configuraciones de vecindades posibles de radio 1/2

- La cantidad de funciones posibles es $2^4 = 16$.

En un AC elemental de radio 1/2 existen 3 funciones interesantes que representan a las 3 compuertas lógicas básicas.

- La regla (8,1/2) ó compuerta lógica AND.

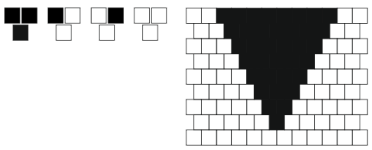

Figura 16: Compuerta AND en autómata celular

- La regla (14,1/2) ó compuerta lógica OR.

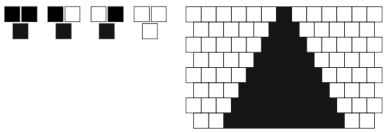

Figura 17: Compuerta OR en autómata celular

- La regla (6,1/2) ó compuerta lógica XOR (Caótica).

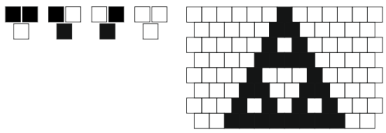

Figura 18: Compuerta XOR en autómata celular

3.1.3. Antecedentes de la aplicación de los autómatas celulares a los campos finitos

En las últimas décadas los autómatas celulares han recibido bastante atención por sus aplicaciones a diversas áreas del conocimiento, como física, química, biología y mas recientemente destacan aplicaciones de cálculo en aritmética de campos finitos. A continuación se presentan dos trabajos que muestran el potencial de cálculo de un AC para realizar operaciones en campos finitos, en especifico los campos de Galois sobre los cuales están definidas las primitivas de los algoritmos criptográficos.

An Evolutionary Approach to the Design of Cellular Automata Architecture for Multiplication in Elliptic Curve Cryptography over Finite Fields [15].

Este trabajo fue desarrollado en la universidad nacional de Kyungpook en Korea por Jun-Cheol Jeon y Kee-Young Yoo, quienes propusieron desarrollar una arquitectura de multiplicación especial y eficiente, basado en un autómata celular para ser aplicado a la criptografía de curvas elípticas sobre el campos finitos de Galois GF(2^n). La arquitectura de computación

evolutiva propuesta fue diseñada e implementada en un dispositivo de hardware con un co-procesador dedicado al cálculo de ECC obteniendo resultados con alta regularidad y con una latencia reducida [15].

En la criptografía de clave publica, para lograr un nivel razonable de seguridad, muchos algoritmos basan sus cálculos en campos de Galois del orden GF(2^{2000}). Por lo tanto hay una necesidad de de desarrollar un algoritmo eficiente para la multiplicación en GF(2^n). Sin embargo parámetros significativamente mas pequeños se suelen utilizar para criptografía de curva elíptica a diferencia de otros algoritmos como El gammal ó RSA cuyos llaves llegan a contar con claves de hasta 100 bytes, Algunos de los beneficios de contar con tamaños de claves mas pequeños incluyen cáculos mas rápidos, y reducción de la potencia de procesamiento, espacio de almacenamiento y ancho de banda [15].

En ECC, la computación kP es la operación más importante, donde k es un número entero y P es un punto de la curva elíptica. Esta operación puede ser calculado usando la adición de dos puntos k veces. Para la realización de dicha operaciones los autores proponen la siguiente arquitectura basada en AC de frontera periódica (PBCA) usando trinomios irreducibles por el metodo MSB-first sobre el campo finito GF(2^n) [15].

Sean $A(x) y B(x) \in GF(2^n)$ y $P(x)$ un trinomio irreducible. la multiplicación $A(X)B(X)$ mod $P(x)$ se desarrolla como sigue:

$$\{[A(x)B_{n-1}x^{n-2}\ mod\ P(x) + A(X)B_{n-2}]x\ mod\ P(x) + ... + A(x)P(x)\}\ mod\ P(x).$$

De la ecuación anterior los autores descomponen las operaciones $R(x)x\ mod\ P(x)$ y $A(x)B_i$ $0 \leq i \leq n-1$ en 3 operaciones mas primitivas para ser implementadas en forma de 3 autómatas celulares, dichos cálculos son: [15]

C1 Shift en $R(x)x$

C2 Reducción modular $R(x)x$ mod $P(x)$

C3 $A(x)B_i\ 0 \leq i \leq n-1$

En primer lugar, con el fin de realizar C1, que requiere un 1 bit de desplazamiento a la izquierda para poner en práctica $R(x)x$, se utiliza autómatas celulares con una frontera periódica usando n-bit de registro. El siguiente estado de cada registro se define como el estado de la vecino de la derecha, la función de transición queda expresada como $Q_i(t+1)=Q_{i-1}(t)$. En este caso, el registro a la izquierda y a la derecha registro son las células adyacentes [15].

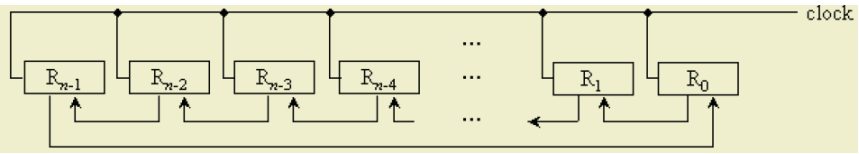

Figura 19: PBCA con regla de transición $Q_i(t+1)=Q_{i-1}(t)$. Figura tomada de [15]

Con el fin de realizar C2, que es la reducción modular, los autores se requieren operaciones de reducción modulares a la izquierda de 1 bit de desplazamiento resultante de C1. La siguiente ecuación se obtiene $R(x)x$ mod $P(x)$ [15].

$$((R_{n-1} \wedge P_{n-1}) \oplus R_{n-2})x^{n-1} + ((R_{n-1} \wedge P_{n-2}) \oplus R_{n-3})x^{n-2} + \ldots + ((R_{n-1} \wedge P_k) \oplus R_{k-1})x^k + \ldots + ((R_{n-1} \wedge P_2) \oplus R_1)x^2 + ((R_{n-1} \wedge P_1) \oplus R_0)x^1 + ((R_{n-1} \wedge P_0) \oplus 0)$$

En la ecuación anterior, $P_i(0 \leq i \leq n-1)$ tiene valores de cero pero Pk y P0 siempre contienen un 1 ya que sólo se tiene en cuenta trinomio polinomio irreductible. La ecuación resultante por C1 y C2 es.

$$R_{n-2}\cdot x^{n-1} + R_{n-3}\cdot x^{n-2} + \ldots + (R_{n-1} \oplus R_{k-1})x^k + \ldots + R_1\cdot x^2 + R_0\cdot x^1 + R_{n-1},$$

Donde $P(x) = x^n + x^k + 1$, la siguiente figura muestra un PBCA que aplica la reducción modular.

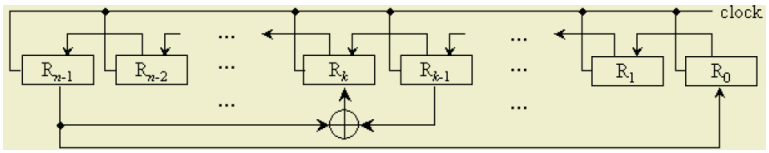

Figura 20: PBCA que aplica $R(x)x$ mod $P(x)$. Figura tomada de [15]

C3 se puede conseguir fácilmente usando n compuertas AND, ya que cada elemento de $A(x)$ debe multiplicarse por el elemento de Bi, donde i esta en el rango $(0 \leq i \leq n-1)$ con el fin de realizar C3 [15].

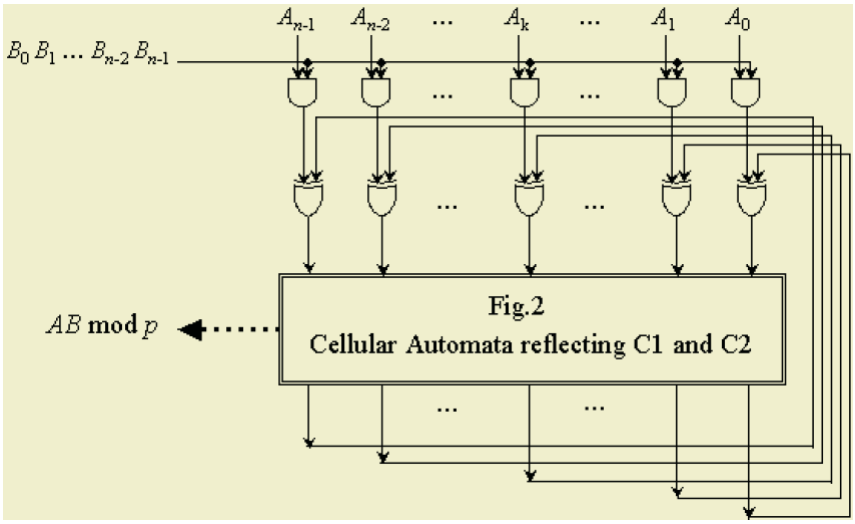

Figura 21: PBCA que aplica C1, C2 y C3. Figura tomada de [15]

A Programmable Parallel Structure to perform Galois Field Exponentiation [16]

En este trabajo se propone una nueva arquitectura de exponenciación en GF(2^m). El núcleo de la arquitectura es una estructura paralela para la multiplicación y la elevación al cuadrado, que se basa en transiciones de estados de un autómata celular. El rendimiento del diseño superó las arquitecturas existentes basadas en arreglos simbólicos por lo que es posible aplicarla de manera efectiva en sistemas criptográficos de clave pública como ElGamal y Diffie-Hellman. Además la naturaleza de los autómatas celulares conduce un diseño VLSI escalable [16].

Los criptosistemas de clave publica como Elgamal, Diffie-Hellman se basan en la exponenciación modular en campos de Galois y aunque operación provee características deseables estas resultan altamente costosas en términos de implementaciones de manera que el rendimiento queda en función de la implementación de la multiplicación [16]. El esquema sobre el cual se basa el diseño la arquitectura es el paralelismo de los algoritmos para calcular multiplicación y el cuadrado, y el hecho de existen coincidencias entre los cálculos de ambos algoritmos, de esta manera los autores presentan un esquema en el cual tanto el cuadrado como la multiplicación re usan el mismo espacio de memoria que contiene los cálculos realizados por algunos de los algoritmos, a dicha estructura la denominan Autómata Celular Programable.

Por cada iteración, los módulos que implementan las operaciones de cuadrados y multiplicación generan una matriz característica del AC, dicho los módulos se ejecutan por separado

operando sobre una semilla que da como resultado una evaluación paralela del producto o cuadrado. El costo de la multiplicación en hardware resulta de $(m + m^2)$ compuertas XOR, $2m^2$ switch's y un ciclo de reloj para evaluar el resultado.

Arquitectura propuesta usando autómata celular:
La arquitectura exponenciación fue implementada en una plataforma de Xilinx XCV-1000 [16].

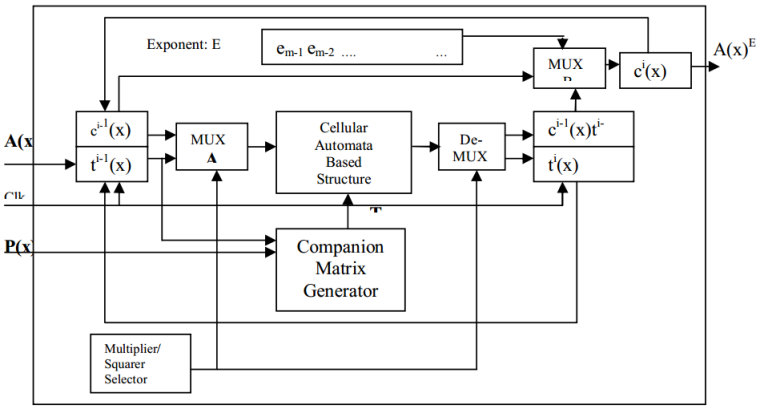

Figura 22: Arquitectura para la exponenciación en GF(2^8). Figura tomada de [16]

El exponente E se codifica en un registro de m-bits $(e_m, e_{m-2}, ..., e_0)$. Al restablecer la entrada A(x) esta es cargada en el registro $t^{i-1}(x)$ y se establece C^{i-1} en 1. El módulo que se encargara de generar la matriz, toma la $t^{i-1}(x)$ entrada y el polinomio primitivo $p(x)$ y calcula la matriz de transición $T_{t,p}$ para el AC. Cuando el AC se configura como multiplicador la linea que controla el selector Cuadrado/Multiplicación hacia el Multiplexor A y el Demultiplexor se establece en 1 permitiendo que C^{i-1} sirva como entrad del AC. Dependiendo del valor menos significativo del exponente e_0 el valor de $c^i(x)$ se establece en el producto computado por el AC (si es 1) o se establece en el valor $c^{i-1}(x)$ (si es 0) a través de Mux B. Finalmente los valores de $c^i(x)$ y $t^i(x)$ es retro alimentado a los registros $c^{i-1}(x)$ y $t^{i-1}(x)$, después de m ciclos de reloj el resultado es cargado al registro $c^i(x)$.

Análisis y Diseño

4. Análisis y Diseño

4.1. Planteamiento del problema

Un autómata celular es un sistema dinámico paralelo cuyo comportamiento esta especificado en términos de una relación local, dichos modelos han sido exitosamente aplicados al diseño de algoritmos paralelos y distribuidos con el objetivo de resolver problemas de sincronización, más recientemente han sido aplicados al computo en campos finitos, específicamente los campos de Galois. En este trabajo terminal se propone caracterizar un algoritmo criptográfico basado en campos de Galois como un sistema dinámico en función de sus operaciones básicas con el objetivo de sustituirlas por un autómata celular que realice dichos cálculos.

El principal problema de caracterizar un algoritmo criptográfico es la obtención de la función de transición, la manera usual de obtener la función de transición es abstraer la dinámica del sistema caracterizando los actores en términos de estados y los cambios que estos sufren a lo largo del tiempo como transiciones de este modo se haya una combinación entre estados y transiciones que describen de manera general al sistema en términos de una función, sin embargo tratándose de un algoritmo criptográfico esta opción no es factible ya que el numero de combinaciones a analizar es muy extenso debido a que en un principio estos son diseñados para ser resistentes a ataques de fuerza bruta. Por lo que es necesario abordar el problema desde otro punto de vista, existen modelos basados en autómata celular que realizan operaciones de computación en campos finitos, por ejemplo existen AC capaces de calcular potencias de un numero en un anillo entero algunos otros realizan operaciones en campo de Galois con el objetivo de ser aplicados a códigos lineales detectores de errores. Sabiendo esto y que existen algoritmos criptográficos cuyas operaciones básicas se definen sobre la teoría de campos finitos, es factible hallar un autómata celular capaz de sustituirlas en las transformaciones de cifrado/descifrado, por lo que es necesario descomponerlo en términos de sus operaciones mas primitivas para ser implementadas en forma de AC.

A continuación se presentan los algoritmos mencionados en el marco teórico que basan sus transformaciones de cifrado/descifrado en campos finitos, con el objetivo de discriminar aquellos cuyas características no se adecuen a los parámetros que se establecerán en el estudio comparativo.

4.1.1. Estudio comparativo

Existen diseños basados en autómatas celulares que permiten realizar cómputos en campos finitos del tipo GF(2^m). Por lo tanto, es posible desarrollar una aplicación de los AC al cifrado de datos. Para poder llevar a cabo dicha aplicación es necesario conocer y comparar los algoritmos que basan su operaciones de cifrado/descifrado en campos finitos del tipo GF(2^m), en específico el campo GF(2^8) que es al que pertenecen las polinomios binarios de grado menor o igual a 7, es decir las cadenas binarias de longitud 8, sobre las cuales se basan las primitivas de algunos de los algoritmos de criptografía.

Para poder realizar dicho estudio comparativo es necesario definir una lista de atributos deseables que debe cumplir un algoritmo para ser implementado en forma de un autómata celular:

1. **Tipo de llave.** Es la pieza información que controla la operación de un algoritmo criptográfico, existen dos tipos de algoritmos respecto a su clave, los de clave pública que consiste de dos claves (una pública y una privada) y los de clave privada, este atributo es importante en el diseño del AC ya que permitirá establecer los argumentos con el que deberá ser inicializado el AC.

2. **Unidad de cifrado.** Los algoritmos criptográficos pueden dividirse con base a la unidad de cifrado, existen cifradores de bloques y de flujo. Los cifradores de bloques son

aquellos que realizan sus operaciones de transformación sobre palabras de una longitud determinada (bloque de información) mientras que los cifradores de flujo realizan sus transformaciones unidad por unidad de información (flujo de datos). El parámetro de unidad de cifrado es relevante en el diseño de la aplicación a nivel conceptual del AC ya que permitirá establecer la longitud de palabra sobre la que el AC deberá operar.

3. **Tamaño del mensaje.** Es la longitud de la unidad de cifrado sin embargo esta puede variar dependiendo del algoritmo ya existen tamaños de mensaje de hasta 64, 128, 256 bits para un mismo algoritmo. Un tamaño de mensaje 128 es deseable por representación en el campo finito GF(2^8).

4. **Tamaño de la clave.** Es la longitud de posibles claves que puede poseer un algoritmo que permite ofrecer un nivel de seguridad en razón a un tamaño de unidad de cifrado. Un tamaño de clave de 128 bits en razón a un tamaño de unidad de cifrado de 128 bits es deseable por representación en el campo finito GF(2^8).

5. **Principal Operación.** Es la operación fundamental en las transformaciones que realiza el algoritmo, esta operación depende del diseño del algoritmo criptográfico ya que diferentes algoritmos basan su cifrado/descifrado en operaciones matemáticas de distinta naturaleza. Una operación principal en un campo GF(2^8) es deseable.

6. **Número de rondas.** Número de ciclos determinados a la que es sometida una unidad de cifrado, esta depende del diseño del algoritmo ya que esta agrega o disminuye el nivel de seguridad según el tamaño de unidad de cifrado y longitud de clave. Esta característica es relevante ya permitirá establecer el numero de veces que el AC se ejecute sobre la unidad de cifrado.

7. **Naturaleza paralela.** (Sin tomar en cuenta modos de operación). Es la factibilidad de paralelización del las operaciones del algoritmo. Esta característica es relevante ya que a diferencia de la naturaleza secuencial de varios algoritmos, el paralelismo es deseable por facilidad de implementación en forma de un AC.

A continuación se mencionan los algoritmo criptográficos candidatos, dando a conocer en una tabla comparativa con el fin de seleccionar el más adecuado de acuerdo a sus características, para implementarlo con un autómata celular.

Propiedades	DES	AES	Twofish	Safer	Elliptic Curves
Unidad de cifrado	Bloques	Bloques	Bloques	Bloques	Secuencial
Tipo de clave	Privada	Privada	Privada	Privada	Pública
Tamaño de mensaje en razón 128	128	128	128	64	128
Tipo de clave en razón 128	64	128	256	64	2(128)=256
Principal operación	permutación y cajas de sustitución	suma, producto y reducción en campo finito	XOR y cajas de sustitución	mezclado de bytes (XOR)	potencia modular
Número de rondas	16	10	16	8	No aplica
Naturaleza paralea	Si	Si	Si	Si	No

Cuadro 5: Tabla comparativa de algoritmos

Según los criterios establecidos en la tabla anterior se ha acordado que el algoritmo que más se adecua a los atributos deseables y necesarios para la realización del proyecto es el algoritmo AES por su descripción matemática en términos de operaciones en campos finitos de Galois. Por lo que será el algoritmo a ser implementado en forma de autómata celular.

Aclaración: En particular para la realización del trabajo terminal únicamente se caracterizará la versión de AES para un bloque de 128 bits, y una clave de 128 bits.

4.1.2. Especificación de AES

AES es el nuevo estándar de cifrado simétrico dispuesto por el NIST, después de un periodo de competencia entre 15 algoritmos sometidos. El 2 de Octubre de 2000 fue designado el algoritmo Rijndael como AES, el estándar reemplazó de TDES.

El algoritmo Rijndael fue elegido por el NIST (National Institute of Standards and Technology), para ser el estándar en los próximos 20 años y es llamado AES (Advanced Encryption Standar). Rijndael fue elegido después de pasar un periodo de análisis durante aproximadamente 3 años, Rijndael fue elegido como la mejor opción dentro de 15 candidatos, sus principales características fueron su fácil diseño, su versatilidad en ser implementado en diferentes dispositivos, soportar bloques de datos de 128 bits y claves de 128, 192, y 256 bits.

Las operaciones en AES están definidas al nivel de bytes, entidades de 8 bits, tratados como elementos del **campo finito** $\mathbb{F}_{2^8}$. [17]

AES es un sistema de cifrado simétrico de bloques que puede procesar los bloques de datos de 128 bits, utilizando claves de cifrado con longitudes de 128, 192 y 256 bits.

Para el algoritmo AES, la longitud del bloque de entrada y el bloque de salida es de 128 bits. la longitud de la clave de cifrado, K, es 128, 192, o 256 bits. La longitud de la clave está representado por Nk = 4, 6, o 8, que refleja el número de palabras de 32 bits (número de columnas) en la clave de cifrado. El número de rondas a ser realizadas durante la ejecución del algoritmo depende en el tamaño de clave. [18]

	Longitud de clave	Tam. de Bloque	No. de Rondas
AES-128	**4**	**4**	**10**
AES-192	6	4	12
AES-256	8	4	14

Cuadro 6: Cuadro de especificación de claves, bloques y rondas

Entradas y salidas

La entrada y la salida para el algoritmo AES consisten cada una en secuencias de 128 bits. Estas secuencias se conocen como bloques y el número de bits que contienen se conocen como su longitud. AES depende de un parámetro extra la clave de cifrado que consiste en secuencia de 128, 192 o 256 bits.

La unidad básica para el procesamiento en el algoritmo AES es un byte, una secuencia de ocho bits tratada como una sola entidad. Las secuencias de entrada, de salida y de cifrado de bits clave se procesan como matrices de bytes que se forman dividiendo estas secuencias en 32 para formar grupos de ocho bits contiguos y obtener matrices de bytes.

Todo byte en AES puede ser representado como un vector de bits individuales, siguiendo el orden $(b_7, b_6, b_5, b_4 b_3, b_2, b_2, b_1, b_0)$. Cada byte es interpretado como un elemento del campo finito GF(2^8), para todo elemento de GF(2^8) tenemos su representación polinomial [18].

$$\sum_{i=0}^{7} b_i x^i = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 1$$

Arreglo de Bytes: un arreglo de bytes puede ser representado como una secuencia de la siguiente forma.

$a_0 a_1 ... a_{15}$ donde cada a_i es un byte

A nivel interno, las operaciones del algoritmo AES se realizan en una matriz bidimensional de bytes llamado el estado. El estado está formado por cuatro filas de bytes, cada una con 4 bytes. En el conjunto de estados, cada S representa un byte individual que tiene dos índices, con su número de renglón r en el rango de 0$\leq$ r $<$ 4 y su número de la columna c en el rango 0$\leq$ c $<$ 4. Esto permite que un byte individual pueda ser referido como $S_{r,c}$ [18].

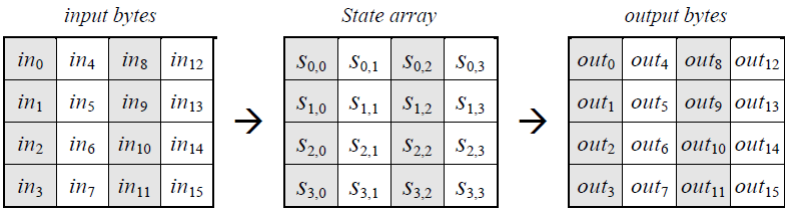

Figura 23: Entrada y salida de un arreglo de estados en AES

Los cuatro bytes en cada columna de la matriz de forman un vector de 32 bits, donde el número de fila r proporciona un índice para los cuatro bytes dentro de cada palabra. El estado por lo tanto, se puede interpretar como una matriz unidimensional de palabras de 32 bits (columnas), $w0...w3$, donde el número de columna c proporciona un índice en esta matriz. Por lo tanto, para la matriz de la figura anterior, el estado puede ser considerado como una matriz de cuatro columnas de 4 bytes, de la siguiente manera [17]:

$$[S_0]=\begin{bmatrix} S_{0,0} \\ S_{1,0} \\ S_{2,0} \\ S_{3,0} \end{bmatrix}, [S_1]=\begin{bmatrix} S_{0,1} \\ S_{1,1} \\ S_{2,1} \\ S_{3,1} \end{bmatrix}, [S_2]=\begin{bmatrix} S_{0,2} \\ S_{1,2} \\ S_{2,2} \\ S_{3,2} \end{bmatrix}, [S_3]=\begin{bmatrix} S_{0,3} \\ S_{1,3} \\ S_{2,3} \\ S_{3,3} \end{bmatrix}$$

4.1.3. Estructura matemática de AES

Una de características mas sobresalientes de AES, comparado con otros algoritmos, es la la estructura matemática sobre la cual opera y esta diseñado, esta característica permite realizar un análisis de cierto modo más fácil respecto a otros algoritmos.

Suma y producto de bytes en AES

Todos los bytes en el algoritmo AES se interpretan como elementos de campos finitos y por lo tanto las operaciones de suma y producto estan dadas de la siguiente manera.

Suma: la adición de dos elementos en un campo finito se logra llevando a cabo con la operación XOR entre dos bytes.

Producto: en la representación polinómica, la multiplicación en GF(2^8) (denotado por •) se corresponde con la multiplicación de polinomios módulo un polinomio irreducible de grado 8. Un polinomio es irreducible si sus únicos divisores son uno y el mismo. Para el algoritmo AES, este polinomio irreducible es $m(x) = x^8+x^4+x^3+1$. La reducción modular de $m(x)$ se asegura de que el resultado será un polinomio binario de grado menor que 8, y por lo tanto puede ser representado por un byte. A diferencia de la suma no hay ninguna operación simple a nivel de byte que corresponde a esta multiplicación. Sin embargo existen algoritmos de cálculo de producto entre elementos del campo finito expresados en combinaciones de operaciones a nivel de bytes, por ejemplo **algoritmo 3** [17].

Inverso multiplicativo en GF(2^8)

Como la multiplicación entre elementos del campo GF(2^8) es asociativa donde el elemento $(01)_{hex}$ es el neutro multiplicativo y por ser GF$(2^8)_{/m(x)}$ campo. Para cualquier polinomio distinto de cero binario $f(x)$ de grado menor que 8, Existe el inverso multiplicativo de $f(x)$, denotado $f^{-1}(x)$, se puede calcular mediante el algoritmo extendido de Euclides para polinomios si se encuentran los polinomios $a(x)$ y $c(x) \in$ GF$(2^8)_{/m(x)}$ tal que [18].

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

Entonces $a(x) \bullet f(x)$ mod $m(x) = 1$ y por lo tanto $f^{-1}(x) = a(x)$.

Definición del producto entre columnas vector con entradas en GF(2^8)

Dado que AES opera con palabras de 32 bytes, podemos considerar que cada palabra es un fila de 4 bytes, así que cada palabra es tratada como un polinomio de grado 3 con coeficientes en GF(2^8) [17].

$(B_3, B_2, B_1, B_0) \rightarrow B_3z^3 + B_2z^2 + B_1z + B_0$ tal que $B_i \in$ GF(2^8).

La multiplicación de dos palabras de 4 bytes puede resultar en un polinomio de grado menor o igual a 6, entonces es necesario reducirlo módulo un polinomio de grado 4 para que el producto quede bien definido. En el algoritmo AES se especifica que el polinomio es:

$$z^4 + 1$$

Aunque el polinomio $z^4 + 1$ no es irreducible, en este caso particular el polinomio constante si tiene inverso en el anillo por lo tanto, ahora trabajaremos en el anillo GF$(2^8)_{/z^4+1}$. Este polinomio tiene la siguiente propiedad útil [19]:

z^i mod $z^4 + 1$ = $z^{i \ mod \ 4}$

DEFINICIÓN 19 (PRODUCTO $\otimes$)
Sean $A(z)$ y $B(z)$ polinomios tales que $grad(A(z)) \leq 3$ y $grad(B(z)) \leq 3$. y sea $M(z) = z^4 + 1$ definimos [18] :

$$A(z) \otimes B(z) = A(z)B(z) \bmod z^4 + 1$$

El objetivo de lo anterior es poder definir multiplicación entre columnas de 4 bytes por un elemento constante también de 4 bytes. Como $A(z), B(z) \in \text{GF}(2^8)_{/z^4+1}$ polinomios tal que $a(z) = A_3z^3 + A_2z^2 + A_1z + a_0$ y $B(z) = B_3z^3 + B_2z^2 + B_1z + B_0$, entonces el producto $A(z)B(z)$ tiene la forma [17]:

$$\begin{aligned} A(z)B(z) &= (A_3z^3 + A_2z^2 + A_1z + A_0)(B_3z^3 + B_2z^2 + B_1z + B_0) \\ &= A_0 \bullet B_0 + (A_1 \bullet B_0 + A_0 \bullet B_1)z \\ &\quad +(A_2 \bullet B_0 + A_1 \bullet B_1 + A_0 \bullet B_2)z^2 \\ &\quad +(A_3 \bullet B_0 + A_2 \bullet B_1 + A_1 \bullet B_2 + A_0 \bullet B_3)z^3 \\ &\quad +(A_3 \bullet B_1 + A_2 \bullet B_2 + A_1 \bullet B_3)z^4 \\ &\quad +(A_3 \bullet B_2 + A_2 \bullet B_3)z^4 + (A_3 \bullet B_3)z^6 \end{aligned}$$

Ahora es necesario aplicar el módulo $x^4 + 1$ al polinomio, que por **algoritmo 2** equivale a aplicar módulo a cada uno de sus términos, entonces $z^i \bmod z^4 + 1 = z^{i \ mod \ 4}$ en cada termino [18].

$$\begin{aligned} A(x) \otimes B(x) &= A_0 \bullet B_0 + A_3 \bullet B_1 + A_2 \bullet B_2 + A_1 \bullet B_3 \\ &\quad +(A_1 \bullet B_0 + A_0 \bullet B_1 + A_3 \bullet B_2 + A_2 \bullet B_3)z \\ &\quad +(A_2 \bullet B_0 + A_1 \bullet B_1 + A_0 \bullet B_2 + A_0 \bullet B_3)z^2 \\ &\quad +(A_3 \bullet B_0 + A_2 \bullet B_1 + A_1 \bullet B_2 + A_0 \bullet B_3)z^3 \\ C(x) &= C_0 + C_1z + C_2z^2 + C_3z^3 \end{aligned}$$

En ecuaciones matriciales queda expresado como: $A(x) \otimes B(x) = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} A_0 & A_3 & A_2 & A_1 \\ A_1 & A_0 & A_3 & A_2 \\ A_2 & A_1 & A_0 & A_3 \\ A_3 & A_2 & A_1 & A_0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$

4.1.4. Cifrado

Para cifrar un bloque con una clave se hacen Nr rondas, en cada una de ellas intervienen 4 transformaciones. Al inicio de la Cifra, la entrada se copia en la matriz de estados. Después de una adición con la clave, la matriz de estados es sometida a cada transformación de manera secuencial durante un lapso de 10, 12 o 14 rondas (dependiendo de la longitud de la clave), a excepción de la ronda final que difieren ligeramente de los primeros Nr-1 rondas ya que no se aplica la última transformación a la matriz de estados. Cada ronda depende de una calendarización de claves que consiste en una matriz de una dimensión 4x4 bytes (en el caso de una clave de 128 bytes) que se obtiene al aplicar una rutina de expansión a la clave inicial, la expansión trabaja sobre dos elementos globales W y Rcon cuyo significado se explica en la rutina de expansión de claves [18].

Algoritmo 4 Cifrado AES con clave de 128 bits

Entrada: Byte TextPlano [4 x 4], Byte K [4 x 4].
Salida: Byte Cifra [4 x 4].

```
1: Byte K[4x4].
2: Byte Estado ← TextoPlano
3: Entero Nr.
4: AddRoundKey(Estado, K)
5: para Nr = 1 hasta 9 hacer
6:    SubBytes(Estado)
7:    ShiftRows(Estado)
8:    MixColumns(Estado)
9:    K ← KeySchendule(K,Nr)
10:   AddRoundKey(Estado, K, Nr)
11: fin para
12: SubBytes(Estado)
13: ShiftRows(Estado)
14: AddRoundKey(Estado, K)
15: devolver Cifra ← Estado
```

En la figura 24 se puede observar las transformaciones a la que es sometida la matriz estado hasta alcanzar el cifrado y como entra en juego la generación de subclaves que resultan de una función de expansión aplicada por cada ronda.

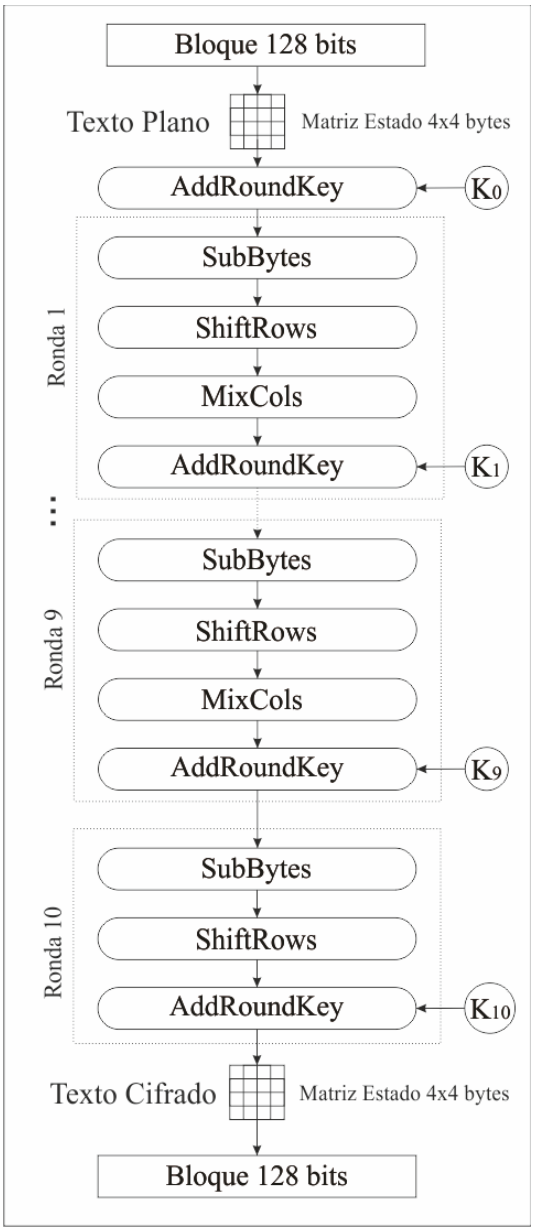

Figura 24: Diagrama de flujo del cifrado AES

AddRoundKey

Esta transformación toma una matriz y simplemente hace un XOR byte a byte de las columnas de la matriz estado con las columnas correspondiente matriz de claves, que depende de la ronda, el bloque resultante de esta transformación, será la nueva matriz estado para la siguiente ronda.

DEFINICIÓN 20 (ADDROUNDKEY)
Sea $[S_j]$ la j-ésima columna de la matriz de estados S y $[K_j]$ la j-ésima columna de la matriz clave K correspondiente la ronda actual, se define AddRoundKey como:

$$\text{AddRoundKey}([S_j])=[S_j]\oplus[K_j].$$

La figura 25 ilustra el modo en el que opera esta transformación con las columnas de la matriz estado con los de la clave de la ronda actual.

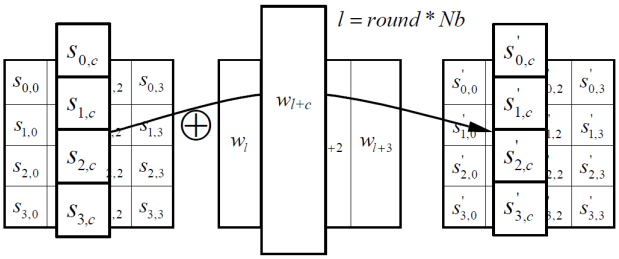

Figura 25: Diagrama de flujo del cifrado AES

SubBytes

Sustitución de bytes es una transformación no lineal que consiste en en sustituir a cada byte de la matriz estado por un nuevo byte, dicha asignación es la que resulta de aplicar las siguientes transformaciones. Como cada byte en AES es considerado un elemento en el campo finito GF$(2^8)_{/x^8+x^4+x^3+1}$, implica que cada elemento tiene inverso multiplicativo a excepción del cero que no tiene inverso, por lo que que se deja inalterado. La idea de la transformación SubBytes es la de extender las funciones de un cifrador affin hacia los polinomios mediante el calculo del inverso multiplicativo de un elemento del campo GF(2^8), el cual se obtiene aplicando el **algoritmo extendido de Euclides para polinomios** para después aplicarle una transformación Affin [17].

DEFINICIÓN 21 (CAJA DE SUSTITUCIÓN S_{box})
Sean $a(x) \neq (00)_{hex}$ y $a^{-1}(x) = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ el inverso de $a(x)$ en GF(2^8) y C la matriz circulante inducida por el vector (10001111). Se define la siguiente transformación Affin sobre $a(x)$.

$$S_{box}(a(x)) = S_{box}((b'_7, b'_6, b'_5, b'_4, b'_3, b'_2, b'_1, b'_0)) = C(a(x))^T + [(C6)_{hex}]^T$$

Por lo que cada b'_i queda expresado de la siguiente forma

$$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

Para 0≤i<8, donde el b_i es el i-ésimo bit del byte, y c_i i-ésimo byte de $(63)_{hex}$ = (01100011). En forma matricial la transformación se puede expresar como:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Transformación Affin S_{box}

Para la transformación Affin S_{box} existen hasta 256 posibles valores, para los cuales se ha calculado todos los posibles valores y se han encuadrado en una tabla denominada Tabla de sustitución, útil en el proceso de cifrado. Gracias a esta tabla aplicar la transformación SubBytes resulta trivial. Consiste en dividir el byte de la matriz estado en dos partes, la parte alta X (los 4 bits mas significativos) y la parte baja Y (los 4 bits menos significativos). Por lo que

Hex		Y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
X	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Cuadro 7: Tabla S-box para la sustitución de bytes de la forma $(XY)_{hex}$.

el valor en la tabla para la fila con el valor X y la columna con el valor Y es el resultado de aplicar la transformación SubBytes al byte $(XY)_{hex}$ [18]. La figura 26 ilustra como la transformación SubBytes debe ser aplicada byte a byte por cada elemento de la matriz de estado, es decir que en la matriz estado la entrada $[S_{i,j}]$ = SubBytes($[S_{i,j}]$), con 0≤i<4, 0≤j<4.

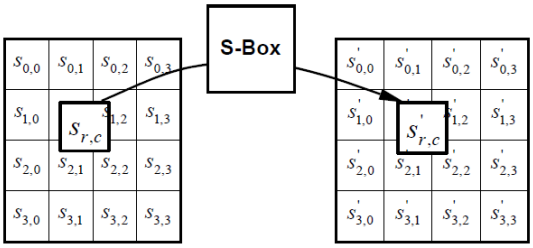

Figura 26: Transformación SubBytes

ShiftRows

En la transformación ShiftRows, los bytes de las ultimas tres filas de la matriz estado son cíclicamente desplazados hacia la izquierda, a cada renglón le es aplicado un diferente offset de desplazamiento. El desplazamiento de los renglones se define de la siguiente manera.

DEFINICIÓN 22 (SHIFTROWS)
Para toda entrada $[S_{i,j}]$ de la matriz de estados $[S_{i,j}]$, se define:

$$\text{ShiftRows}:[S_{i,j}] \rightarrow [S_{i,(j-i) \ mod \ 4}] \text{ con } 0 \leq i < 4,\ 0 \leq j < 4.$$

Según la definición anterior cada fila se le aplica un corrimiento circular hacia la derecha un numero determinado de posiciones, la fila 0 se desplaza 0 posiciones, la fila 1 se desplaza 1 posición, la fila 2, dos posiciones y la fila 3, 3 posiciones. La figura 27 ilustra claramente esta transformación.

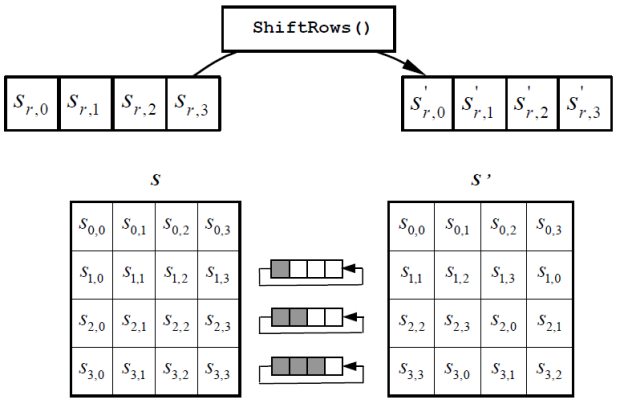

Figura 27: Transformación ShiftRows

MixColumns

La transformación MixColumns opera con vectores columnas de 4 bytes de la matriz estado y las multiplica por un elemento constante de GF$(2^8)_{/z^4+1}$, siguiendo la definición 19 del producto $\otimes$ se define MixColumns [18]:

DEFINICIÓN 23 (MIXCOLUMNS)
Sean [S$_j$] la j-ésima columna de la matriz estado y sea $A(z) = (03)_{hex}x^3+(01)_{hex}x^2+(01)_{hex}x+(02)_{hex} \in$ GF$(2^8)_{/z^4+1}$ un elemento constante, se define MixColumns como:

$$\text{MixColumns}([S_j])=[S_j](z) \otimes A(z)$$

Desarrollando el producto $\otimes$: $A(z)\otimes S_j(z) = \begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} (02)_{hex} & (03)_{hex} & (01)_{hex} & (01)_{hex} \\ (01)_{hex} & (02)_{hex} & (03)_{hex} & (01)_{hex} \\ (01)_{hex} & (01)_{hex} & (02)_{hex} & (03)_{hex} \\ (03)_{hex} & (01)_{hex} & (01)_{hex} & (02)_{hex} \end{bmatrix} \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix}$

Donde $S'_{i,j}$ denota el nuevo valor en la entrada [$S_{i,j}$] de la matriz estado.
[$S_{0,j}$]=$((02)_{hex} \bullet S_{0,j}) \oplus ((03)_{hex} \bullet S_{1,j}) \oplus S_{2,j} \oplus S_{3,j}$
[$S_{1,j}$]=$S_{0,j} \oplus ((02)_{hex} \bullet S_{1,j}) \oplus ((03)_{hex} \bullet S_{2,j}) \oplus S_{3,j}$
[$S_{2,j}$]=$S_{0,j} \oplus S_{1,j} \oplus ((02)_{hex} \bullet S_{2,j}) \oplus ((03)_{hex} \bullet S_{3,j})$
[$S_{3,j}$]=$((03)_{hex} \bullet S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus ((02)_{hex} \bullet S_{3,j})$

Donde $\bullet$ denota el producto en campo finito GF$(2^8)_{/m(x)}$ que se puede calcular de manera sencilla mediante el **algoritmo 3**. La figura 28 ilustra el modo en que opera la transformación MixColumns sobre las columnas la de la matriz estado [18].

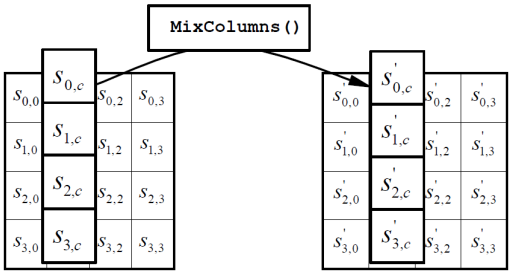

Figura 28: Transformación MixColumns

4.1.5. KeySchendule ó expansión de llaves

Aunque AES esta diseñado para soportar una gran variedad longitudes de claves y de bloques sin embargo el estándar solo permite claves con longitudes de 128, 192 y 256. En la elaboración de este trabajo terminal únicamente se considera claves de de claves 128 bits, aunque el algoritmo puede ser extendidos fácilmente a los casos restantes [19].

La función de expansión genera subclaves a partir de la clave inicial K. Esta función de expansión opera sobre un arreglo lineal W de longitud Nb(Nr+1) de columnas de bytes. Para una clave de 128 bits, Nb=4 y Nr=10 por lo tanto la longitud del arreglo lineal es de 4(11)=44 columnas de bytes, Si se tratara una clave de 192 bits Nb=6 y Nr=12 y para una clave de 256 bits Nb=8 y Nr=14. Las primeras 4 columnas de arreglo corresponden a las primeras 4 columnas de la clave inicial K, mientras que el resto de columnas de bytes se van generando por cada ronda. La figura 29 representa el arreglo W para una clave de 128 bits y de las subclaves generadas por cada ronda que W almacena, donde W_i es una columna de 4 bytes [18].

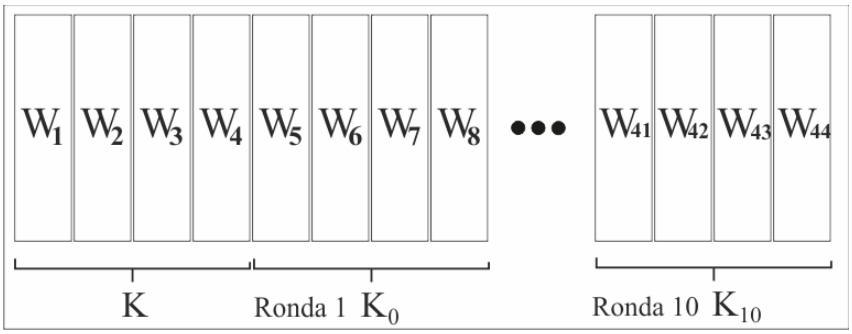

Figura 29: Arreglo W generado en KeySchendule

En la generación de las subclaves interviene un elemento constante denotado por Rcon, que es un arreglo unidimensional de columnas de 4 bytes de longitud Nr-1, para una clave de 128 bits Rcon tiene longitud de 9, Rcon es tal que cada columna de bytes contiene los valores de $[(02)^{Nr-1}_{hex}$,$(00)_{hex}$,$(00)_{hex}$,$(00)_{hex}]$ donde $(02)^{Nr-1}_{hex}$ es la Nr-1 potencia de $(02)_{hex}$ en GF(2^8).La figura 30 muestra el arreglo Rcon para una clave de 128 bits que contiene en cada columna la i-ésima potencia de $[(02)^{Nr-1}_{hex}$,$(00)_{hex}$,$(00)_{hex}$,$(00)_{hex}]$, con 1≤i≤9 [18].

Rcon

[02]	[04]	[08]	[10]	[20]	[40]	[80]	[1B]	[36]
[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]
[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]
[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]	[00]

Figura 30: Arreglo Rcon para una clave de 128 bits

En la expansión de claves intervienen 2 transformaciones a nivel columna.

- **SubWord:** es una función que toma una palabra de entrada de cuatro bytes y le aplica la transformación S_{box} como se había construido anteriormente en la definición 21.
- **RotWord:** toma la j-ésima columna $[W_j]=[W_{0,j},W_{1,j},W_{2,j},W_{3,j}]$ le realiza una permutación cíclica, y devuelve $[W_{1,j},W_{2,j},W_{3,j},W_{0,j}]$, es decir $RotWord([W_{i,j}])=[W_{(i-1)\ mod\ 4,j}]$ [18].

Algoritmo 5 Algoritmo de expansión de claves ó KeySchendule para una clave de 128 bits

Entrada: Byte Key [4x4], Entero Nr.
Salida: Byte K_{Nr} [4x4].

```
Byte T [4X1].
Byte W[4x44].
Entero i=0
mientras i < Nk hacer
    W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3])
    i=i+1
fin mientras
i=Nk
mientras i < 44 hacer
    T = W[i-1]
    si i mod Nr = 0 entonces
        T ← SubWord(RotWord(T)) ⊕ Rcon[i/Nr]
    si no
        si (Nr > 6) ∧ (i mod Nk = 4) entonces
            T ← sSubWord(T)
        fin si
    fin si
    W[i] = W[i-Nr] ⊕ T
    i = i+1
fin mientras
devolver Byte K_Nr[4x4]=(W[Nr*4+1],W[Nr*4+2],W[Nr*4+3],W[Nr*4+4])
```

La siguiente figura muestra un diagrama de la expansión de claves, $K_{i,j}$ representa la i-ésima columna de bytes de la subclave j y K_j la j-ésima subclave correspondiente la ronda j.

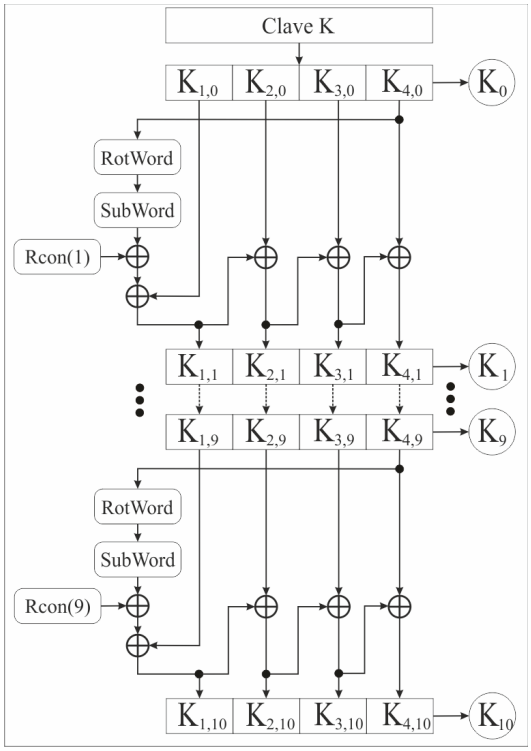

Figura 31: Expansor de claves AES para una clave de 128 bits

4.1.6. Descifrado

El proceso de descifrado es muy similar al del cifrado, solo es necesario realizar todas las operaciones en orden inverso y usar la generación de subclaves, también, en orden inverso, es decir la clave inicial sera la ultima subclave generada.

Algoritmo 6 Descifrado AES con clave de 128 bits

Entrada: Byte Cifra [4 x 4], Byte K [4 x 4].
Salida: Byte Cifra [4 x 4].

```
1: Byte W[4x44].
2: Byte W[4x9].
3: Byte Estado ← Cifra
4: Entero Nr.
5: AddRoundKey(Estado, K)
6: para Nr = 9 hasta 1 step -1 downto 1 hacer
7:    InvShiftRows(Estado)
8:    InvSubBytes(Estado)
9:    K ← KeySchendule(K,Nr)
10:   AddRoundKey(Estado, K, Nr)
11:   InvMixColumns(Estado)
12: fin para
13: InvShiftRows(Estado)
14: InvSubBytes(Estado)
15: AddRoundKey(Estado, K)
16: devolver TextoPlano ← Estado
```

InvShiftRows

La transformación InvShiftRows es la inversa de los bytes en las tres últimas filas del Estado se desplazan cíclicamente durante un número diferente de posiciones, la figura 32 ilustra la forma en que opera InvShiftRows sobre los elementos de la matriz de estados. La posición en la entrada [$S_{r,c}$] se da por la ecuación [18]:

DEFINICIÓN 24 (INVSHIFTROWS)
InvShiftRows([$S'_{r,c}$])=[$S_{r,(c+4-j)\ mod\ 4}$]

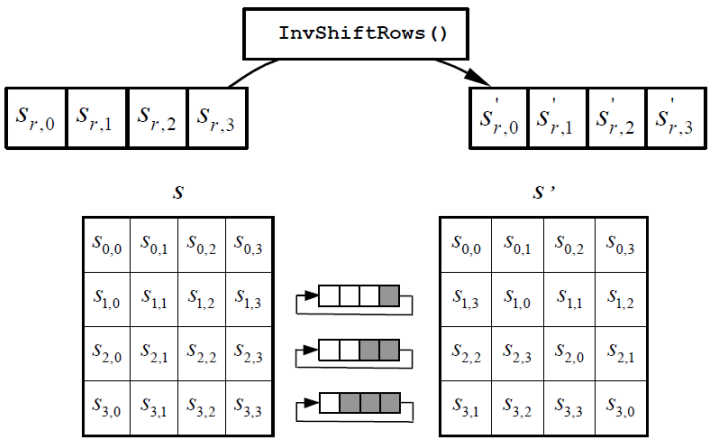

Figura 32: Transformación InvShiftRows

InvSubBytes

InvSubBytes es la inversa de la transformación SubBytes (sustitución de bytes). Sean $b(x) = (b'_7, b'_6, b'_5, b'_4, b'_3, b'_2, b'_1, b'_0)$ = $S_{box}(a(x))$ y C la matriz circulante inducida por el vector por el vector (10001111) define la siguiente transformación Affin sobre $b(x)$ [18].

DEFINICIÓN 25 (INV S-BOX)
$S^{-1}_{box}(b(x)) = (C^{-1}[(b'_7, b'_6, b'_5, b'_4, b'_3, b'_2, b'_1, b'_0)^T + [(C6)_{hex}]^T])^{-1} = a(x)$

Al igual que S_{box}, ya existe una tabla que contiene los 256 valores para S^{-1}_{box}.

Hex		Y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
X	**0**	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	8	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	9	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	A	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	B	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	C	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	D	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	E	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	E	17	2b	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Cuadro 8: Tabla Inv S-box para la sustitución de bytes de la forma $(XY)_{hex}$

InvMixColumns

InvMixColumns es la transformación inversa de MixColumns, ambas operan sobre columnas de 4 bytes las cuales son consideradas polinomios en GF$(2^8)_{/z^4+1}$ y las multiplica por un elemento constante, siguiendo la definición 19 del producto $\otimes$ se define InvMixColumns [18]:

DEFINICIÓN 26 (INVMIXCOLUMNS)
InvMixColumns([S$_j$])=$A^{-1}(z)$[$\otimes$S$_j$](z) con $A^{-1}(z)$=(0B)$_{hex}z^3$+(0D)$_{hex}z^2$+(09)$_{hex}z$+(0E)

Desarrollando el producto $\otimes$:

$$A^{-1}(z) \otimes S_j(z) = \begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} (02)_{hex} & (0E)_{hex} & (0D)_{hex} & (09)_{hex} \\ (01)_{hex} & (09)_{hex} & (0B)_{hex} & (0D)_{hex} \\ (01)_{hex} & (0D)_{hex} & (0E)_{hex} & (0B)_{hex} \\ (03)_{hex} & (0B)_{hex} & (09)_{hex} & (0E)_{hex} \end{bmatrix} \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix}$$

Donde S$'_{i,j}$ denota el nuevo valor en la entrada [S$_{i,j}$] de la matriz estado.

$[S_{0,j}]=((0E)_{hex} \bullet S_{0,j}) \oplus ((0B)_{hex} \bullet S_{1,j}) \oplus ((0D)_{hex} \bullet S_{2,j}) \oplus ((09)_{hex} \bullet S_{3,j})$
$[S_{1,j}]=((09)_{hex} \bullet S_{0,j}) \oplus ((0E)_{hex} \bullet S_{1,j}) \oplus ((0B)_{hex} \bullet S_{2,j}) \oplus ((0D)_{hex} \bullet S_{3,j})$
$[S_{2,j}]=((0D)_{hex} \bullet S_{0,j}) \oplus ((09)_{hex} \bullet S_{1,j}) \oplus ((0E)_{hex} \bullet S_{2,j}) \oplus ((0B)_{hex} \bullet S_{3,j})$
$[S_{3,j}]=((0B)_{hex} \bullet S_{0,j}) \oplus ((0D)_{hex} \bullet S_{1,j}) \oplus ((09)_{hex} \bullet S_{2,j}) \oplus ((0E)_{hex} \bullet S_{3,j})$

Donde $\bullet$ denota el producto en campo finito $GF(2^8)_{/m(x)}$ que se puede calcular de manera sencilla mediante el **Algoritmo 3**.

4.1.7. Seguridad del algoritmo AES

La seguridad de un sistema criptográfico depende generalmente de que al menos una de las claves empleadas sea secreta, más que de que el algoritmo de cifrado sea secreto. El publicar los algoritmos empleados por un sistema criptográfico para que sean revisados públicamente es una buena práctica que permite que se mejoren algoritmos no totalmente seguros o se considere que un algoritmo no tiene debilidades [51], existen 6 principios relativos a las propiedades deseables que un sistema criptográfico que se considere seguro debe cumplir, dichos supuestos se conoces como los principios de Kerckhoffs y dicen [20]:

1. Si el sistema no es teóricamente irrompible, al menos debe serlo en la práctica.
2. La efectividad del sistema no debe depender de que su diseño permanezca en secreto.
3. La efectividad del sistema debe recaer en el secretismo de la clave.
4. Los criptogramas deberán dar resultados alfanuméricos.
5. El sistema debe ser operable por una única persona.
6. El sistema debe ser fácil de utilizar.

Según dicho principio, la seguridad del sistema debe recaer en la seguridad de la clave debiéndose suponer conocidos el resto de los parámetros del sistema criptográfico, por lo que la longitud de la clave es crítica para determinar la susceptibilidad de un cifrador frente a un ataque de búsqueda exhaustiva. Las claves se usan para controlar la operación de un cifrador de manera tal que sólo la clave correcta pueda transformar el texto cifrado a texto plano. Muchos cifradores están basados en algoritmos conocidos públicamente incluido el AES, de manera que la seguridad del sistema depende exclusivamente de la clave [20]. La seguridad de los algoritmos criptográficos poseen distintos grados de seguridad respecto a su clave:

1. Seguro computacionalmente. Con suficiente poder de cálculo y almacenamiento el sistema puede ser roto, pero a un coste tan elevado que no es práctico. De cualquier modo, el coste computacional para considerar que un algoritmo es seguro ha ido cambiando con el paso del tiempo; algoritmos antes considerados seguros, como el DES, han sido rotos en meses con sistemas distribuidos y en días con sistemas diseñados específicamente para la tarea [51].
2. Seguro incondicionalmente. Son aquellos en los que aun disponiendo de recursos y gran cantidad de texto cifrado no es posible romper el algoritmo. Los únicos sistemas incondicionalmente seguros son los one time pads [51].

Un sistema criptográfico puede ser roto en varios niveles [51]:

- Deducción de información. Se obtiene parte de información de la clave o del texto en claro.
- Deducción de una instancia. Se obtiene el texto en claro a partir de un texto cifrado.

- Deducción global. A partir de la deducción de una instancia se obtiene un algoritmo que obtiene los mismos resultados que el algoritmo original.
- Rotura total. Se recupera la clave y se puede descifrar cualquier mensaje cifrado con la misma clave.

Para que un sistema criptográfico sea considerado como fuerte debe tener las siguientes características [51]:

1. Debe disponer de un número muy elevado de claves posibles, de modo que sea poco razonable intentar descifrar un mensaje por el método de la fuerza bruta (probando todas las claves).
2. Debe producir texto cifrado que parezca aleatorio a un test estadístico estándar.
3. Debe resistir todos los métodos conocidos de romper los códigos, es decir, debe ser resistente al criptoanálisis.

La seguridad del algoritmo AES radica en la longitud de su llave, AES ofrece una seguridad de 128, 192 y 256 bits, en este trabajo terminal únicamente abordaremos una seguridad de 128 bits, hasta el momento solo se ha registrado un ataque exitoso a AES en 2011, pero los autores no revelaron información acerca del ataque simplemente lo confirmaron. Cabe mencionar que ademas de ser fuerte ante ataques de fuerza bruta este algoritmo no es vulnerable al criptoanálisis diferencial ni al lineal.

En el cirptoanálisis lineal el atacante conoce o puede adivinar el texto de alguna parte del texto cifrado. La tarea es descifrar el resto del bloque cifrado utilizando esta información. Esto puede ser hecho determinando la clave utilizada para cifrar la información, o a través de algún atajo, es una forma general de criptoanálisis sobre la base de la búsqueda de aproximaciones afines para la acción de un sistema de cifrado. Este tipo de ataques se han especializado en criotgrafía simétrica.

En el criptoanálisis diferencial el atacante puede tener cualquier texto cifrado con una clave desconocida. La tarea es determinar la clave utilizada para cifrar. Es una técnica de tipo estadístico consistente en cifrar parejas de texto en claro escogidas con la condición de que su producto or-exclusivo obedezca a un patrón definido previamente. Los patrones de los correspondientes textos cifrados suministran información con la que conjeturar la clave criptográfica.

Otro tipo de criptoanálsis es el de diccionario, en este ataque no se pretende obtener la clave, sino directamente el texto original a través de la clave. Por una parte, se dispone de un diccionario con gran cantidad de palabras y combinaciones muy comunes y válidas como claves. Posteriormente se realiza su cifrado con la utilidad del sistema a atacar o una copia de la misma, se comprueba el resultado del cifrado con el contenido del archivo de claves y en el caso de que se produzca una coincidencia se infiere cuál es el mensaje sin cifrar.

4.1.8. Ejemplo del algoritmo AES

En esta sección se muestra una prueba de escritorio cifrando una instancia del algoritmo, el mensaje es automatacellular y la clave que se asignaron valores aleatorios. También se muestran los valores intermedios de las funciones del algoritmo (todos los valores están en formato hexadecimal).

Cifrado con una llave de 128 bits.

Esta sección contiene el cifrado de un texto con una clave de una longitud de 128 bits. Las transformaciones son aplicadas al estado durante 10 ciclos, el ciclo final no incluye la función Mixcols.

Mensaje: 61 75 74 6F 6D 61 74 61 63 65 6C 6C 75 6C 61 72

Clave de cifrado: 0f 15 71 c9 47 d9 e8 59 0c b7 ad d6 af 7f 67 98

Ronda	AddRoundKey	SubBytes	ShiftRows	MixColumns	SubClave
Entrada	$\begin{bmatrix} 61 & 6d & 63 & 75 \\ 75 & 61 & 65 & 6c \\ 74 & 74 & 6c & 61 \\ 6f & 61 & 6c & 72 \end{bmatrix}$				$\begin{bmatrix} 0f & 47 & 0c & af \\ 15 & d9 & b7 & 7f \\ 71 & e8 & ad & 67 \\ c9 & 59 & d6 & 98 \end{bmatrix}$
Ronda 1	$\begin{bmatrix} 6e & 2a & 6f & da \\ 60 & b8 & d2 & 13 \\ 05 & 9c & c1 & 06 \\ a6 & 38 & ba & ea \end{bmatrix}$	$\begin{bmatrix} 9f & e5 & a8 & 57 \\ d0 & 6c & b5 & 7d \\ 6b & de & 78 & 6f \\ 24 & 07 & f4 & 87 \end{bmatrix}$	$\begin{bmatrix} 9f & e5 & a8 & 57 \\ 6c & b5 & 7d & d0 \\ 78 & 6f & 6b & de \\ 87 & 24 & 07 & f4 \end{bmatrix}$	$\begin{bmatrix} 6e & 5e & a0 & ef \\ 48 & 01 & e8 & 61 \\ 91 & e2 & 0a & 27 \\ bb & a6 & fb & 04 \end{bmatrix}$	$\begin{bmatrix} dc & 9b & 97 & 38 \\ 90 & 49 & fe & 81 \\ 37 & df & 72 & 15 \\ b0 & e9 & 3f & a7 \end{bmatrix}$
Ronda 2	$\begin{bmatrix} b2 & c5 & 37 & d7 \\ d8 & 48 & 16 & e0 \\ a6 & 3d & 78 & 32 \\ 0b & 4f & c4 & a3 \end{bmatrix}$	$\begin{bmatrix} 37 & a6 & 9a & 0e \\ 61 & 52 & 47 & e1 \\ 24 & 27 & bc & 23 \\ 2b & 84 & 1c & 0a \end{bmatrix}$	$\begin{bmatrix} 37 & a6 & 9a & 0e \\ 52 & 47 & e1 & 61 \\ bc & 23 & 24 & 27 \\ 0a & 2b & 84 & 1c \end{bmatrix}$	$\begin{bmatrix} 2e & 96 & b7 & 84 \\ 46 & 66 & ab & b9 \\ 18 & da & a4 & 05 \\ a3 & c3 & 63 & 6c \end{bmatrix}$	$\begin{bmatrix} d2 & 49 & de & e6 \\ c9 & 80 & 7e & ff \\ 6b & b4 & c6 & d3 \\ b7 & 5e & 61 & c6 \end{bmatrix}$
Ronda 3	$\begin{bmatrix} fc & df & 69 & 62 \\ 8f & e6 & d5 & 46 \\ 73 & 6e & 62 & d6 \\ 14 & 9d & 02 & aa \end{bmatrix}$	$\begin{bmatrix} b0 & 9e & f9 & aa \\ 73 & 8e & 03 & 5a \\ 8f & 9f & aa & f6 \\ fa & 5e & 77 & ac \end{bmatrix}$	$\begin{bmatrix} b0 & 9e & f9 & aa \\ 8e & 03 & 5a & 73 \\ aa & f6 & 8f & 9f \\ ac & fa & 5e & 77 \end{bmatrix}$	$\begin{bmatrix} f4 & 2e & d6 & 32 \\ fe & 63 & 99 & 81 \\ 9e & 7f & 44 & 65 \\ ac & a3 & 79 & e7 \end{bmatrix}$	$\begin{bmatrix} c0 & 89 & 57 & b1 \\ af & 2f & 51 & ae \\ df & 6b & ad & 7e \\ 39 & 67 & 06 & c0 \end{bmatrix}$
Ronda 4	$\begin{bmatrix} 34 & a7 & 81 & 83 \\ 51 & 4c & c8 & 2f \\ 41 & 14 & e9 & 1b \\ 95 & c4 & 7f & 27 \end{bmatrix}$	$\begin{bmatrix} 18 & 5c & 0c & ec \\ d1 & 29 & e8 & 15 \\ 83 & fa & 1e & af \\ 2a & 1c & d2 & cc \end{bmatrix}$	$\begin{bmatrix} 18 & 5c & 0c & ec \\ 29 & e8 & 15 & d1 \\ 1e & af & 83 & fa \\ cc & 2a & 1c & d2 \end{bmatrix}$	$\begin{bmatrix} 99 & 1e & b8 & 83 \\ a4 & 57 & a4 & 92 \\ 42 & 8f & 20 & bf \\ 9c & f7 & ba & bb \end{bmatrix}$	$\begin{bmatrix} 2c & a5 & f2 & 43 \\ 5c & 73 & 22 & 8c \\ 65 & 0e & a3 & dd \\ f1 & 96 & 90 & 50 \end{bmatrix}$
Ronda 5	$\begin{bmatrix} b5 & bb & 4a & c0 \\ f8 & 24 & 86 & 1e \\ 27 & 81 & 83 & 62 \\ 6d & 61 & 2a & eb \end{bmatrix}$	$\begin{bmatrix} d5 & ea & d6 & ba \\ 41 & 36 & 44 & 72 \\ cc & 0c & ec & aa \\ 3c & ef & e5 & e9 \end{bmatrix}$	$\begin{bmatrix} d5 & ea & d6 & ba \\ 36 & 44 & 72 & 41 \\ ec & aa & cc & 0c \\ e9 & 3c & ef & e5 \end{bmatrix}$	$\begin{bmatrix} ee & 95 & 02 & 45 \\ 7f & bb & 92 & c9 \\ 00 & a5 & 0d & d7 \\ 77 & b3 & 1a & 49 \end{bmatrix}$	$\begin{bmatrix} 58 & fd & 0f & 4c \\ 9d & ee & cc & 40 \\ 36 & 38 & 9b & 46 \\ eb & 7d & ed & bd \end{bmatrix}$
Ronda 6	$\begin{bmatrix} b6 & 68 & 0d & 09 \\ e2 & 55 & 5e & 89 \\ 36 & 9d & 96 & 91 \\ 9c & ce & f7 & f4 \end{bmatrix}$	$\begin{bmatrix} 4e & 45 & d7 & 01 \\ 98 & fc & 58 & a7 \\ 05 & 5e & 90 & 81 \\ de & 8b & 68 & bf \end{bmatrix}$	$\begin{bmatrix} 4e & 45 & d7 & 01 \\ fc & 58 & a7 & 98 \\ 90 & 81 & 05 & 5e \\ bf & de & 8b & 68 \end{bmatrix}$	$\begin{bmatrix} ac & 3d & c9 & 87 \\ b9 & b3 & 06 & a0 \\ 53 & 7d & fc & 9d \\ db & b1 & cd & 15 \end{bmatrix}$	$\begin{bmatrix} 71 & 8c & 83 & cf \\ c7 & 29 & e5 & a5 \\ 4c & 74 & ef & a9 \\ c2 & bf & 52 & ef \end{bmatrix}$
Ronda 7	$\begin{bmatrix} dd & b1 & 4a & 48 \\ 7e & 9a & e3 & 05 \\ 1f & 09 & 13 & 34 \\ 19 & 0e & 9f & fa \end{bmatrix}$	$\begin{bmatrix} c1 & c8 & d6 & 52 \\ f3 & b8 & 11 & 6b \\ c0 & 01 & 7d & 18 \\ d4 & ab & db & 2d \end{bmatrix}$	$\begin{bmatrix} c1 & c8 & d6 & 52 \\ b8 & 11 & 6b & f3 \\ 7d & 18 & c0 & 01 \\ 2d & d4 & ab & db \end{bmatrix}$	$\begin{bmatrix} 1a & 74 & 61 & 70 \\ 00 & 16 & f0 & 77 \\ f4 & 8e & c0 & d5 \\ c7 & f9 & 87 & a9 \end{bmatrix}$	$\begin{bmatrix} 37 & bb & 38 & f7 \\ 14 & 3d & d8 & 7d \\ 93 & e7 & 08 & a1 \\ 48 & f7 & a5 & 4a \end{bmatrix}$
Ronda 8	$\begin{bmatrix} 2d & cf & 59 & 87 \\ 14 & 2b & 28 & 0a \\ 67 & 69 & c8 & 74 \\ 8f & 0e & 22 & e3 \end{bmatrix}$	$\begin{bmatrix} d8 & 8a & cb & 17 \\ fa & f1 & 34 & 67 \\ 85 & f9 & e8 & 92 \\ 73 & ab & 93 & 11 \end{bmatrix}$	$\begin{bmatrix} d8 & 8a & cb & 17 \\ f1 & 34 & 67 & fa \\ e8 & 92 & 85 & f9 \\ 11 & 73 & ab & 93 \end{bmatrix}$	$\begin{bmatrix} 5a & b2 & 0a & 51 \\ 13 & 3c & 3a & 7b \\ d1 & 14 & 5b & aa \\ 48 & c5 & e9 & 07 \end{bmatrix}$	$\begin{bmatrix} 48 & f3 & cb & 3c \\ 26 & 1b & c3 & be \\ 45 & a2 & aa & 0b \\ 20 & d7 & 72 & 38 \end{bmatrix}$
Ronda 9	$\begin{bmatrix} 12 & 41 & c1 & 6d \\ 35 & 27 & f9 & c5 \\ 94 & b6 & f1 & a1 \\ 68 & 12 & 9b & 3f \end{bmatrix}$	$\begin{bmatrix} c9 & 83 & 78 & 3c \\ 96 & cc & 99 & a6 \\ 22 & 4e & a1 & 32 \\ 45 & c9 & 14 & 75 \end{bmatrix}$	$\begin{bmatrix} c9 & 83 & 78 & 3c \\ cc & 99 & a6 & 96 \\ a1 & 32 & 22 & 4e \\ 75 & 45 & c9 & 14 \end{bmatrix}$	$\begin{bmatrix} 12 & da & ea & 83 \\ c7 & b9 & 80 & cd \\ c3 & b1 & da & 0a \\ c7 & bf & 85 & b4 \end{bmatrix}$	$\begin{bmatrix} fd & 0e & c5 & f9 \\ 0d & 16 & d5 & 6b \\ 42 & e0 & 4a & 41 \\ cb & 1c & 6e & 56 \end{bmatrix}$
Ronda 10		$\begin{bmatrix} df & 48 & 15 & da \\ 74 & 79 & fc & 24 \\ 0c & d1 & 60 & b3 \\ fe & 0a & e9 & 98 \end{bmatrix}$	$\begin{bmatrix} df & 48 & 15 & da \\ 79 & fc & 24 & 74 \\ 60 & b3 & 0c & d1 \\ 98 & fe & 0a & e9 \end{bmatrix}$		$\begin{bmatrix} b4 & 8e & f3 & 52 \\ ba & 98 & 13 & 4e \\ 7f & 4d & 59 & 20 \\ 86 & 26 & 18 & 76 \end{bmatrix}$
Salida	$\begin{bmatrix} 6b & f2 & 6a & 5c \\ f7 & 64 & 69 & 52 \\ 93 & a0 & 55 & c9 \\ ca & b0 & 2a & 9f \end{bmatrix}$				

Subclaves del cifrado

La siguiente tabla muestra las subclaves de cada ronda así como las funciones intermedias que nos permiten obtener las claves (todos los valores están en formato hexadecimal).

Ronda	Clave	RotWord	SubWord	Rcon
Inicial	$\begin{bmatrix} 0f & 15 & 71 & c9 \\ 47 & d9 & e8 & 59 \\ 0c & b7 & ad & d6 \\ af & 7f & 67 & 98 \end{bmatrix}$	$\begin{bmatrix} 7f \\ 67 \\ 98 \\ af \end{bmatrix}$	$\begin{bmatrix} d2 \\ 85 \\ 46 \\ 79 \end{bmatrix}$	$\begin{bmatrix} d3 \\ 85 \\ 46 \\ 79 \end{bmatrix}$
Ronda 1	$\begin{bmatrix} dc & 90 & 37 & b0 \\ 9b & 49 & df & e9 \\ 97 & fe & 72 & 3f \\ 38 & 81 & 15 & a7 \end{bmatrix}$	$\begin{bmatrix} 81 \\ 15 \\ a7 \\ 38 \end{bmatrix}$	$\begin{bmatrix} 0c \\ 59 \\ 5c \\ 07 \end{bmatrix}$	$\begin{bmatrix} 0e \\ 59 \\ 5c \\ 07 \end{bmatrix}$
Ronda 2	$\begin{bmatrix} d2 & c9 & 6b & b7 \\ 49 & 80 & b4 & 5e \\ de & 7e & c6 & 61 \\ e6 & ff & d3 & c6 \end{bmatrix}$	$\begin{bmatrix} ff \\ d3 \\ c6 \\ e6 \end{bmatrix}$	$\begin{bmatrix} 16 \\ 66 \\ b4 \\ 8e \end{bmatrix}$	$\begin{bmatrix} 12 \\ 66 \\ b4 \\ 8e \end{bmatrix}$
Ronda 3	$\begin{bmatrix} c0 & af & df & 39 \\ 89 & 2f & 6b & 67 \\ 57 & 51 & ad & 06 \\ b1 & ae & 7e & c0 \end{bmatrix}$	$\begin{bmatrix} ae \\ 7e \\ c0 \\ b1 \end{bmatrix}$	$\begin{bmatrix} e4 \\ f3 \\ ba \\ c8 \end{bmatrix}$	$\begin{bmatrix} ec \\ f3 \\ ba \\ c8 \end{bmatrix}$
Ronda 4	$\begin{bmatrix} 2c & a5 & f2 & 43 \\ 5c & 73 & 22 & 8c \\ 65 & 0e & a3 & dd \\ f1 & 96 & 90 & 50 \end{bmatrix}$	$\begin{bmatrix} 8c \\ dd \\ 50 \\ 43 \end{bmatrix}$	$\begin{bmatrix} 64 \\ c1 \\ 53 \\ 1a \end{bmatrix}$	$\begin{bmatrix} 74 \\ c1 \\ 53 \\ 1a \end{bmatrix}$
Ronda 5	$\begin{bmatrix} 58 & 9d & 36 & eb \\ fd & ee & 38 & 7d \\ 0f & cc & 9b & ed \\ 4c & 40 & 46 & bd \end{bmatrix}$	$\begin{bmatrix} 40 \\ 46 \\ bd \\ 4c \end{bmatrix}$	$\begin{bmatrix} 09 \\ 5a \\ 7a \\ 29 \end{bmatrix}$	$\begin{bmatrix} 29 \\ 5a \\ 7a \\ 29 \end{bmatrix}$
Ronda 6	$\begin{bmatrix} 71 & c7 & 4c & c2 \\ 8c & 29 & 74 & bf \\ 83 & e5 & ef & 52 \\ cf & a5 & a9 & ef \end{bmatrix}$	$\begin{bmatrix} a5 \\ a9 \\ ef \\ cf \end{bmatrix}$	$\begin{bmatrix} 06 \\ d3 \\ df \\ 8a \end{bmatrix}$	$\begin{bmatrix} 46 \\ d3 \\ df \\ 8a \end{bmatrix}$
Ronda 7	$\begin{bmatrix} 37 & 14 & 93 & 48 \\ bb & 3d & e7 & f7 \\ 38 & d8 & 08 & a5 \\ f7 & 7d & a1 & 4a \end{bmatrix}$	$\begin{bmatrix} 7d \\ a1 \\ 4a \\ f7 \end{bmatrix}$	$\begin{bmatrix} ff \\ 32 \\ d6 \\ 68 \end{bmatrix}$	$\begin{bmatrix} 7f \\ 32 \\ d6 \\ 68 \end{bmatrix}$
Ronda 8	$\begin{bmatrix} 48 & 26 & 45 & 20 \\ f3 & 1b & a2 & d7 \\ cb & c3 & aa & 72 \\ 3c & be & 0b & 38 \end{bmatrix}$	$\begin{bmatrix} be \\ 0b \\ 38 \\ 3c \end{bmatrix}$	$\begin{bmatrix} ae \\ 2b \\ 07 \\ eb \end{bmatrix}$	$\begin{bmatrix} b5 \\ 2b \\ 07 \\ eb \end{bmatrix}$
Ronda 9	$\begin{bmatrix} fd & 0d & 42 & cb \\ 0e & 16 & e0 & 1c \\ c5 & d5 & 4a & 6e \\ f9 & 6b & 41 & 56 \end{bmatrix}$	$\begin{bmatrix} 6b \\ 41 \\ 56 \\ f9 \end{bmatrix}$	$\begin{bmatrix} 7f \\ 83 \\ b1 \\ 99 \end{bmatrix}$	$\begin{bmatrix} 49 \\ 83 \\ b1 \\ 99 \end{bmatrix}$
Ronda 10	$\begin{bmatrix} b4 & 8e & f3 & 52 \\ ba & 98 & 13 & 4e \\ 7f & 4d & 59 & 20 \\ 86 & 26 & 18 & 76 \end{bmatrix}$			

4.2. Análisis

El análisis y la especificación de requisitos es una tarea de ingeniería del software que cubre el hueco entre la definición del software a nivel sistema y el diseño de software. El análisis de requerimientos permite especificar las características operacionales del software, indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software [21].

4.3. Diseño

El diseño es el proceso de varios pasos en el cual las representaciones de la estructura de los datos y el programa, las características de la información y el detalle procedimental se sintetizan a partir de los requisitos [21]. En este apartado nos preocupamos en abstraer la representación del sistema, su estructura y las secuencias del procesamiento, además de los componentes necesarios para construir el sistema y sus características.

Dentro de las funcionalidades que el sistema debe cumplir, destacan por su nivel de prioridad las de cifrado con AES, descifrado con AES, cifrado con AC y descifrado con AC por lo que a continuación se presenta el diseño conceptual de dichas funcionalidades.

4.3.1. Cifrar/Descifrar con AES

Debido a que AES un cifrador de bloques sus transformaciones de cifrado/descifrado operan en unidades de cifrado que constan de grupos de bits de longitud 128, llamados bloques. AES toma un bloque de texto plano como entrada al cual aplica 10 rondas de transformaciones para producir un bloque de igual tamaño conocido como cifra la figura 33 muestra el proceso de cifrado/descifrado que realiza AES en una única unidad de cifrado.

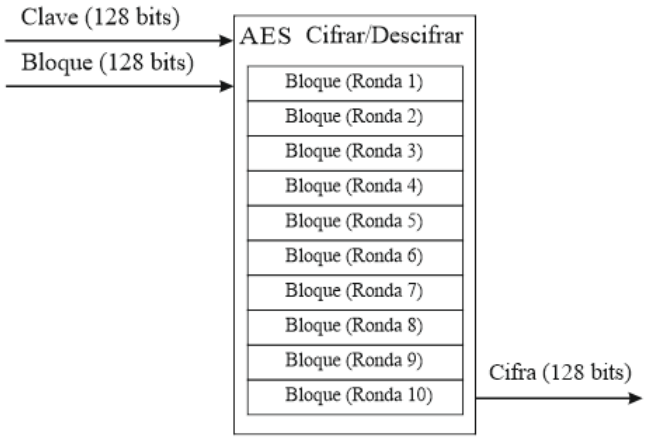

Figura 33: Proceso de cifrado/descifrado realizado por AES en un solo bloque.

Cuando se desea cifrar un mensaje que supera los 128 bits de tamaño, se procede a descomponer el mensaje en bloques secuenciales de 128 bits, a cada bloque le es aplicado el proceso descrito con anterioridad hasta obtener las cifras independientes de cada bloque, luego se unen las cifras para formar un texto cifrado, la imagen 33 ilustra este proceso, cabe destacar que este proceso considera que cada bloque debe ser cifrado con la misma clave.

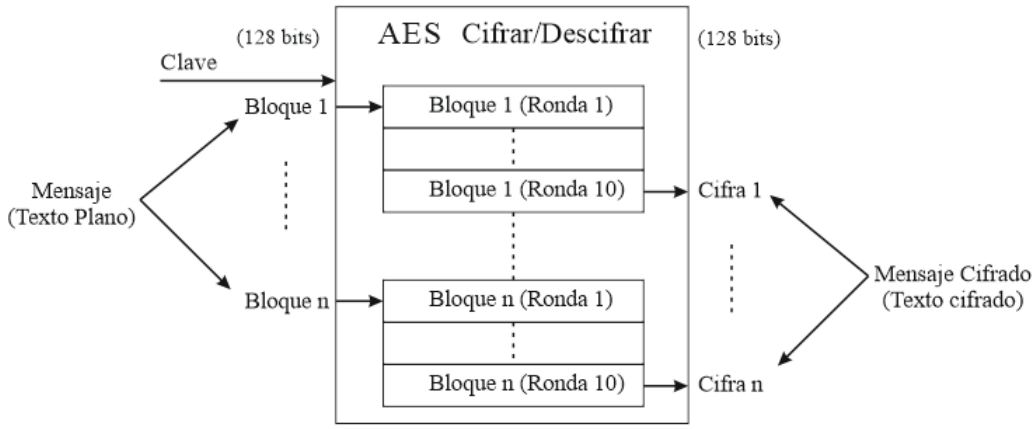

Figura 34: Proceso de cifrado/descifrado realizado por AES a múltiples bloques que conforman un mensaje.

4.3.2. Cifrar/Descifrar con autómata celular

Como el objetivo principal del trabajo terminal es caracterizar un algoritmo criptográfico (en este caso AES) en forma de un autómata celular, con el fin de implementar las transformaciones de cifrado y descifrado en forma de un AC, es necesario establecer conceptualmente los elementos que intervienen en dicho proceso. En primer lugar como el algoritmo caracterizado es un cifrador de bloques el AC debe cifrar por bloques, esto se obtiene segmentado el mensaje en unidades de cifrado a las cuales se les aplicaran reglas de evolución que emulan las operaciones básicas sobre las cuales esta basado AES (suma y producto en GF(2^8)), sin embargo tratar el bloque de información en su forma habitual resulta inconveniente en el momento de definir la regla de evolución, haciendo necesaria una codificación del bloque de información en un bloque de estados que el AC pueda calcular, de igual manera este mismo proceso debe ser aplicada a la clave privada. Una vez obtenida la codificación, el AC aplica las reglas de evolución de manera reiterativa haciendo evolucionar la cinta de estados hasta obtener una cifra equivalente a la del algoritmo AES, la cifra obtenida por el AC en primera instancia se encuentra en términos de estados así que para obtener la cifra equivalente a la producida por el algoritmo AES es necesario realizar la codificación inversa a la original, la imagen 35 refleja claramente este proceso en una sola unidad de cifrado mientras que la imagen 34 muestra el proceso aplicado a múltiples bloques que conforman un mensaje, la codificación y su justificación se especifica más adelante en el apartado que contiene la obtención de las reglas.

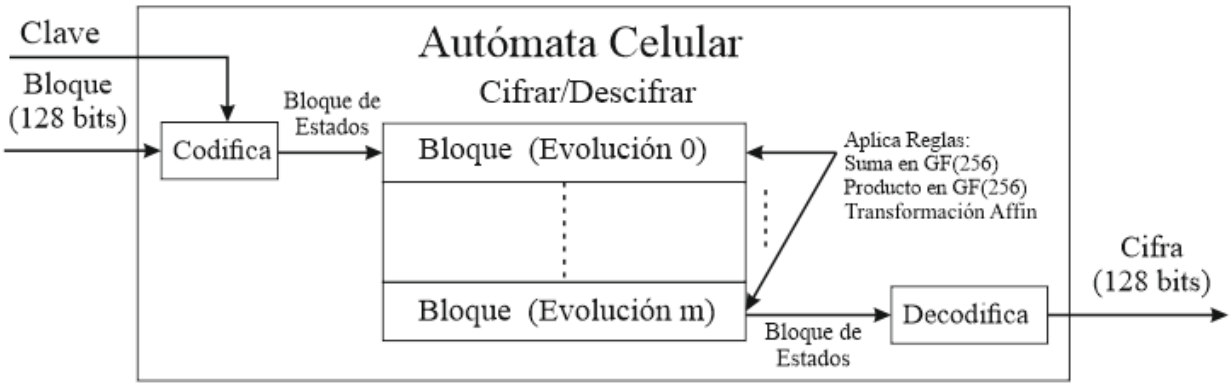

Figura 35: Proceso de cifrado/descifrado con AC en una sola unidad de cifrado (bloque)

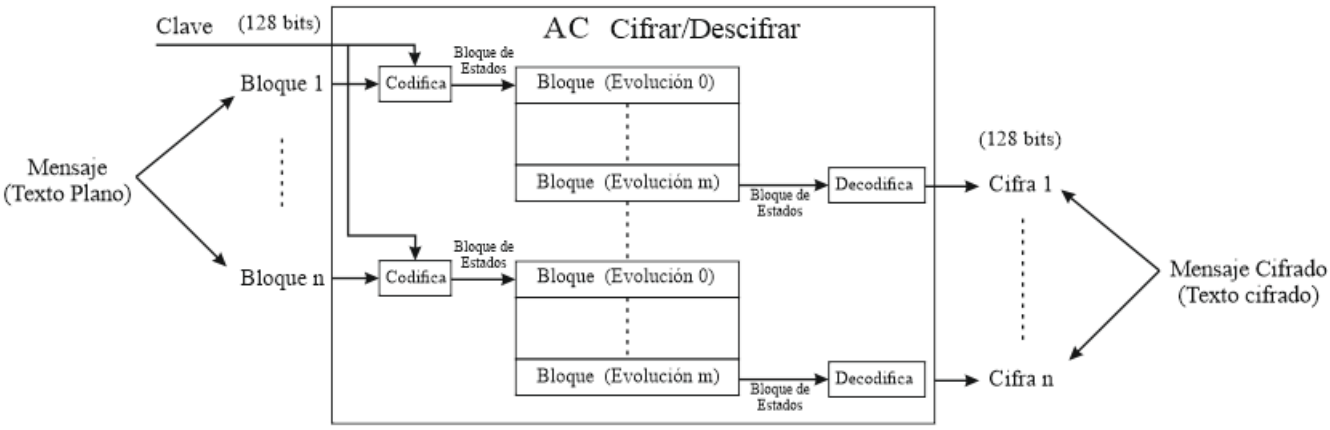

Figura 36: Proceso de cifrado/descifrado realizado por AC a múltiples bloques que conforman un mensaje.

4.3.3. Diagrama de clases

Descripción de clases

- **MensajeTexto:** es la entidad encargada de poseer los métodos y atributos relacionados con el archivo a transformar.
- **Primitivas (interface):** Es la entidad encargada de especificar las primitivas criptográficas cifrado/descifrado que AC y AES deben realizar.
- **AES:** es la entidad encargada de proveer los servicios criptográficos cifrado/descifrado a un mensaje, implementando la forma clásica de cifrado establecida en el estándar del algoritmo AES.
- **Expansor de claves:** el la entidad encargada de expandir una clave y proveer subclaves por cada ronda solicitada.
- **Autómata celular cifrador:** es la entidad encargada de proveer los servicios criptográficos cifrado/descifrado a un mensaje, implementando operaciones en forma de autómata celular.
- **Reticula:** es la entidad encargada de contener un arreglo de estados que forma parte de la definición de un autómata celular.
- **Lienzo:** es la entidad encargada de proveer visualización de los cálculos realizados por el AC.
- **Codificador de bloques:** es la unidad encargada de preparar la información para cifrado/descifrado conformando bloques de estados ó polinomios.

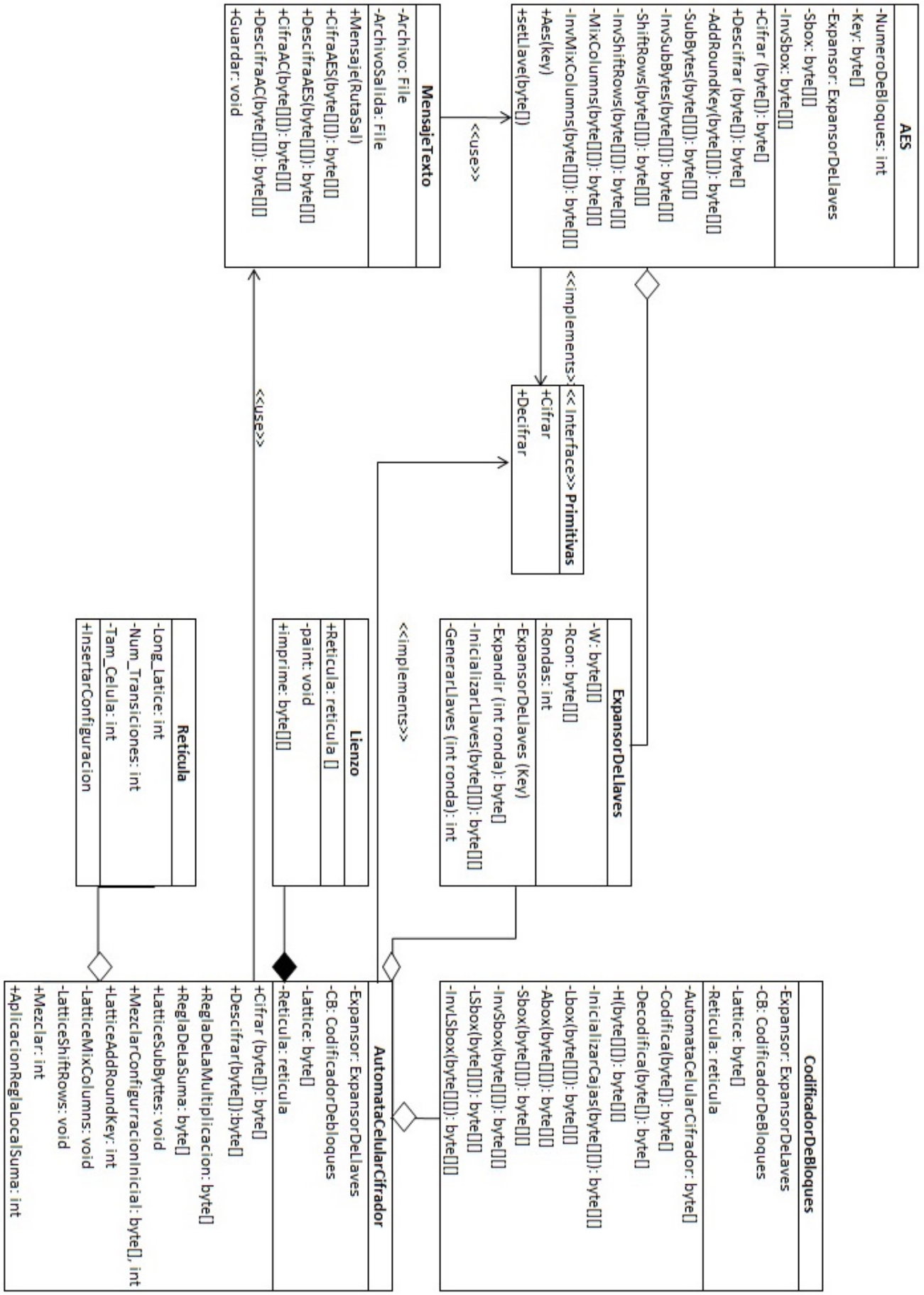

Figura 37: Diagrama de clases

4.4. Especificación del caso de estudio

Al largo del presente trabajo se ha trabajado con el concepto de algoritmo sin embargo Donald Knut introduce una definición más formal, el concibe a un algoritmo como una secuencia finita de instrucciones, reglas o pasos que describen de modo preciso las operaciones que una computadora debe realizar para ejecutar una tarea determinada en un tiempo finito. En la práctica, un algoritmo es un método para resolver problemas mediante los pasos o etapas siguientes:

1. Diseño del algoritmo: expresar el algoritmo como un programa en un lenguaje de programación adecuado (fase de codificación). Las dos herramientas más comúnmente utilizadas para diseñar algoritmos son diagramas de flujo y pseudocódigos [22].

2. Ejecución y validación del programa por la computadora: el resultado final del diseño es una solución que debe ser fácil de traducir a estructuras de datos y estructuras de control de un lenguaje de programación específico [22].

Una vez dispongamos de un algoritmo que funciona correctamente, en este caso el algoritmo AES para el cifrado de datos, es necesario definir criterios para medir su rendimiento o comportamiento. Estos criterios se centran principalmente en su simplicidad y en el uso de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo. Dichos parámetros van a servir además para comparar algoritmos entre sí, permitiendo determinar el más adecuado de entre varios que solucionan un mismo problema. En este trabajo terminal nos centraremos solamente en el comportamiento temporal.

El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que le suministremos, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo. Hay dos estudios posibles sobre el tiempo:

1. Uno que proporciona una medida teórica (a priori), que consiste en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados.

2. Y otro que ofrece una medida real (a posteriori), que consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en una computadora concreta.

Ambas medidas son importantes puesto que, si bien la primera nos ofrece estimaciones del comportamiento de los algoritmos de forma independiente de la computadora en donde serán implementados y sin necesidad de ejecutarlos, la segunda representa las medidas reales del comportamiento del algoritmo. Estas medidas son funciones temporales de los datos de entrada.

4.4.1. Complejidad algorítmica de AES

Una complejidad algorítmica es una medida teórica a priori que permite estimar el costo de un algoritmo, para este trabajo terminal únicamente nos enfocaremos en la complejidad temporal. Para poder obtener la complejidad del algoritmo AES es necesario especificar que el tamaño de la entrada de un algoritmo es el número de componentes sobre los que se va a ejecutar el algoritmo en este caso el tamaño del bloque. La unidad de tiempo a la que debe hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra

unidad de tiempo concreta, pues no existe una computadora estándar al que puedan hacer referencia todas las medidas. Denotaremos por $T(n)$ el tiempo de ejecución de un algoritmo para una entrada de tamaño n. Teóricamente $T(n)$ debe indicar el número de instrucciones ejecutadas por una computadora idealizada. Debemos buscar por tanto medidas simples y abstractas, independientes de la máquina a utilizar. Para ello es necesario acotar de alguna forma la diferencia que se puede producir entre distintas implementaciones de un mismo algoritmo, ya sea del mismo código ejecutado por dos máquinas de distinta velocidad, como de dos códigos que implementen el mismo método. Esta diferencia es la que acota el principio de invarianza que asegura que el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa.

Cuando describimos cómo es que el número de operaciones $T(n)$ depende del tamaño n; lo que nos interesa es encontrar el patrón de crecimiento para la función de complejidad y así caracterizar al algoritmo; una vez hecha esta caracterización podremos agrupar las funciones de acuerdo al número de operaciones que realizan. Para tal propósito utilizaremos la notación de orden $O(T(n))$ que representa el conjunto de funciones que son dominadas asintóticamente dominadas por $T(n)$ [52].

De acuerdo a los conceptos enunciados, debe resultar claro que el análisis de un algoritmo precisa de un conteo de los recursos consumidos. En un algoritmo iterativo como lo es AES, una vez identificadas las operaciones básicas y ciclos se procede a realizar el análisis, se realiza utilizando técnicas tradicionales de conteo, partiendo de los siguientes supuestos:

1. Las sentencias simples, aquellas que tienen que ver con la asignación, operaciones matemáticas básicas, entrada/salida, etc. Siempre y cuando no trabajen sobre variables estructuradas cuyo tamaño este relacionado con el tamaño del problema (lectura o escritura de arreglos, archivos, puertos). Requieren un tiempo constante de ejecución, siendo su complejidad O(1) [60].

2. Los condicionales suele ser O(1), a menos que estos involucren un llamado a un procedimiento, y siempre se le debe sumar la peor complejidad posible de las alternativas del condicional [60].

3. En los ciclos con un contador explícito, se distinguen dos casos: cuando el tamaño n forma parte de los límites del ciclo, con una complejidad basada en n ó bien cuando el ciclo se realiza un número constante de veces, independiente de n por lo que la repetición sólo introduce una constante multiplicativa que puede absorberse, dando como resultado O(1) [60].

Análisis del algoritmo AES

El algoritmo 4 presenta el pseudocódigo del cifrado realizado por AES para una clave de 128 bits, el siguiente análisis aplicado al algoritmo AES permitirá obtener una medida a priori sobre la complejidad del algoritmo AES en su versión de 128 bits. Para ellos es necesario calcular la complejidad de cada una de las transformaciones que lo componen.

- **Complejidad de AddRoundKey:** esta transformación consiste en una compuerta XOR elemento con elemento entre las matrices de estados y la matriz clave, cabe mencionar que no es necesario recorrer la matrices con dos contadores pues el tamaño es fijo para el AES de 128 bits ya que solo son necesarias 16 Compuertas XOR, 16 asignaciones 4 sentencias de decisión simples (if) y 4 operaciones módulo para desplazarse sobre los renglones con lo que solo basta con un contador explicito 16 posiciones, por lo tanto para un bloque de tamaño n, $T_{ARK}(n)$ para AddroundKey es $T(n) = 16+16+8 = 40$ con $T_{ARK}(n) = O(1)$.

- **Complejidad de SubBytes:** esta transformación consiste en una substitución de bytes de la forma byte (XY) mediante la tabla 8, donde la parte alta X es un renglón y la parte baja Y le corresponde a una columna, dicha sustitución resulta trivial pues no es necesario realizar una búsqueda sobre la tabla 8 ya que al byte (XY) le corresponde de manera univoca una entrada determinada por los valores de X y Y por lo que se requieren 1 asignación y 1 referencia la tabla en función de los dos nibbles del byte resultando un total de 4 operaciones por sustitución, ahora debemos tomar en cuenta que la matriz exige 16 sustituciones de este tipo de modo que $T_{SB}(n) = 16(4)$ con $T_{SB}(n) = O(1)$.

- **Complejidad de ShiftRows:** Esta transformación consiste en una permutación de los elementos de la matriz, que tiene la particularidad de que la entrada con coordenadas (i, j) se desplazara a la posición $(i, (j - i)\ mod\ 4)$, notemos que el primer renglón no se desplaza, el segundo se desplaza en forma de anillo una posición a la izquierda, el tercero dos posiciones a la izquierda y el cuarto tres posiciones la izquierda, por lo que solo son necesarios 12 intercambios, por lo tanto ShiftRows tiene una complejidad $T_{SR}(n) = 12$ con $T_{SR}(n) = O(1)$.

- **Complejidad de MixColumns:** esta transformación consiste en un producto de matrices, entre la matriz de estados y una matriz constante para la cual existe inversa con entradas del campo GF(2^8). Esta operación es la más costosa puesto que una columna de la matriz de estados involucra 16 compuertas XOR, 4 asignaciones al vector columna y 16 productos en campo finito ($\bullet$), sin embargo como el producto $\bullet$ es una operación binaria para la cual existen diversas implementaciones la complejidad del producto de una columna por una matriz queda en función de la complejidad de $\bullet$ a la cual denotaremos por $T_{\bullet}(m)$, donde m es el grado máximo o de alguno los dos polinomios a multiplicarse es decir $m = Max\{grad(p(x)), grad(q(x))\}$. Así que en un producto de dos matrices de bytes esta dada por $T_{MC}(n) = 4(4 + 16 + 16(T_{\bullet}(m))) = 80 + 4T_{\bullet}(m)$, con lo que la complejidad de MixColumns es $O(n) = max\{O(1), O_{\bullet}(m)\}$.

Como ya se conocen las complejidades de cada transformación es posible calcular la complejidad del cifrado AES, observemos que las primeras tres lineas corresponden a sentencias simples con complejidad contante $O(1)$, la cuarta linea es una llamada a AddRoundKey con complejidad $O(1)$, después las lineas 6 hasta las 10 entran en un ciclo con un contador explicito desde 1 hasta 9, dichas lineas contienen las llamadas a las 4 transformaciones y las lineas desde la 12 hasta la 13 contienen tres llamadas a 3 transformaciones con complejidad $O(1)$. Así podemos decir que la complejidad del cifrado AES vienen dada por la siguiente expresión.

$$T_{AES}(n) = T(n)+T_{ARK}(n)+9(T_{SB}(n)+T_{SR}(n)+T_{MC}(n)+T_{ARK}(n))+T_{SB}(n)+T_{SR}(n)+T_{ARK}(n).$$

$$T_{AES}(n) = K_1 + K_2 T_{MC}(n)\ con\ K_1\ y\ K_2\ constantes.$$

Como se puede observar la transformación MixColumns es la única que no posee una complejidad constante, por lo que la complejidad de el algoritmo AES queda en función de la complejidad del algoritmo utilizado para realizar producto en campo finito GF(2^n) y por lo tanto.

$$T_{AES}(n) = Max\{O(1), O_{\bullet}(T_{bullet}(m))\}.$$

Si el producto en GF(2^8) ($\bullet$) se implementa haciendo uso de **Algoritmo 3**, la complejidad de AES esta dada por.

$$T_{AES}(n) = O(n)$$

La complejidad del descifrado es la misma que la del cifrado puesto que implementan las mismas operaciones pero con parámetros y orden diferente.

Caracterización de las operaciones primitivas

5. Caracterización

5.1. Caracterización de operaciones primitivas

La caracterización del algoritmo AES, consiste en encontrar un conjunto de estados y funciones de transición de un autómata celular que puedan ser mapeados mediante una codificación a elementos del campo y operaciones sobre las cuales el algoritmo esta construido, de este modo los elementos del campos se volverán estados, las operaciones de campo funciones de transición y las estructura de datos matriz será una estructura de datos lattice.

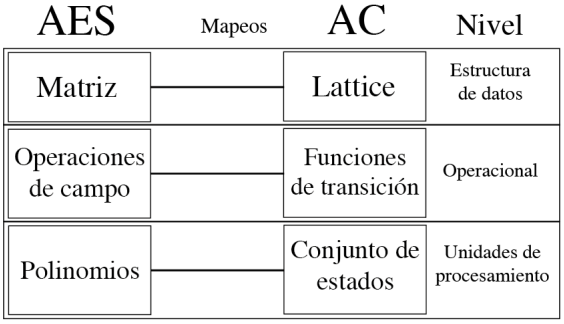

Figura 38: Diagrama que muestra los mapeos entre ambas entidades.

En el algoritmo AES existen tres operaciones primitivas a ser caracterizadas, que transforman directamente las unidades de procesamiento, y son las siguientes:

1. El producto en campo finito.
2. La transformación affin de elementos del campo finito.
3. La suma en campo finito.

5.1.1. Autómata celular multiplicador

Consideremos el conjunto $\mathbb{Z}_{23} = \{0, 1, 2, ..., 22\}$ con el objeto de diseñar un AC que permita realizar multiplicaciones módulo 23, para luego extender la construcción al campo de los polinomios GF(2^8). En general cuando se trata de campos finitos modelar la multiplicación en su correspondiente tabla de multiplicar resulta poco factible debido a que existen una gran cantidad posibles productos, por lo que buscar patrones sobre la tabla para logar encontrar la función de transición no resulta viable. Sin embargo los campos finitos poseen propiedades interesantes que facilitan su análisis, entre ellas existe la propiedad de contener un subconjunto de elementos denominado conjunto primitivo, cuyos elementos tienen la capacidad de generar al campo finito, dichos elementos reciben el nombre de generadores ó generatrices.

DEFINICIÓN 27 (ELEMENTO GENERADOR)
Sea $\mathbb{F}_p$ un campo finito entonces se cumple que $\exists$ e $\in \mathbb{F}_p$ elemento distinguido y un $n \in \mathbb{N}$ tal que $\forall$ a $\in \mathbb{F}_p$ $a = e^n$, e se denomina generador de $\mathbb{F}_p$.

$$\mathbb{F}_p = \{a | a = e^n \; con \; n \in \mathbb{N} \; y \; a \; \neq 0\}$$

La idea de citar la propiedad anterior es la de encontrar una tabla que permita codificar los elementos de un campo en términos de las potencias enteras de algún elemento generador, para así poder obtener función de transición totalística que caracterice el producto modular. Retomando $\mathbb{Z}_{23}$, como 23 es número primo entonces $\mathbb{Z}_{23}$ es campo finito. Se sabe por una

búsqueda exhaustiva que 5 es elemento generador de $\mathbb{Z}_{23}$, con lo que se conforma la tabla 9 de potencias de 5, notemos que por ser generador la tabla no solo es colectivamente exhaustiva si no que también el generador induce una permutación sobre el orden natural de aparición de los elementos de $\mathbb{Z}_{23}$, ahora el orden que se tomará sobre los elementos del cuerpo sera el del exponente. Una observación importante es que el 0 a pesar de ser elemento del cuerpo este no puede ser generado ya que los campos finitos cumplen con la propiedad de ser dominio de integridad, es decir que dados dos elementos distintos de cero el producto de estos siempre es distinto de cero aunque se traten de módulos, es por eso que el 0 no figurará en la siguiente tabla.

Exponente	Potencia	Exponente	Potencia	Exponente	Potencia	Exponente	Potencia
0	1	5	20	11	22	17	15
1	5	6	8	12	18	18	16
2	2	7	17	13	21	19	7
3	10	8	16	14	13	20	12
4	4	9	11	15	19	21	14

Cuadro 9: Potencias de base 5 módulo 23

El objetivo principal de este análisis es la de encontrar una forma de hacer que el AC tome dos elementos de $\mathbb{Z}_{23}$ dispuestos en células adyacentes y calcule su producto en la siguiente transición, sin embargo el problema radica en trabajar los elementos del campo directamente, pues resulta inconveniente por la cantidad de operaciones aritméticas que involucra un producto modular, por lo que es necesario buscar una codificación univoca de $\mathbb{Z}_{23}$ que permita expresar el producto del campo en una función de transición totalística. La codificación ideal es la generada por la tabla 9 pues notemos que los exponentes también son todos elementos de $\mathbb{Z}_{23} - \{0\}$, así que existe una función biyectiva entre los $\mathbb{Z}_{23} - \{0\}$ a los elementos de $\mathbb{Z}_{23} - \{0\}$, a dicha función la denominaremos codificación que tomará una potencia de 5 módulo 23 y le asignara su correspondiente exponente, por ser la tabla una función biyectiva entonces tiene inversa a la que le denominamos decodificación, esta función inversa toma un número entre 0 y 22, y le asigna una potencia de 5 módulo 23. La codificación es casi completa, el único detalle es que como el cero no puede ser generado este no figura en la tabla aunque sea parte del cuerpo, esto se arregla con agregar una entrada más a la tabla donde el exponente es Z con potencia 0, a la nueva tabla le llamaremos tabla de codificación de estados base 5, por el hecho que toma un valor y le asigna una codificación sobre la cual trabajara el AC es decir un estado.

Estado	Z	0	1	2	3	4	5	6	7	8	9	10
Valor	0	1	5	2	10	4	20	8	17	16	11	9
Estado	11	12	13	14	15	16	17	18	19	20	21	
Valor	22	18	21	13	19	3	15	6	7	12	14	

Cuadro 10: Tabla de codificación de estados base 5.

Sabemos que si e_1 y e_2 son estados de la tabla 10 entonces tenemos dos casos el primero cuando ambos estados son diferentes de Z, $e_1 \neq Z$ y $e_2 \neq Z$ lo que implica que $e_1 = n$ y $e_2 = m$ tales que 5^n, $5^m \in \mathbb{Z}_{23}$, si desarrollamos el producto $(5^n)(5^m)$ tenemos que:

$$(5^n)(5^m) \text{ mod } 23 = 5^{(n+m)} \text{ mod } 23 = 5^{(e_1+e_2)} \text{ mod } 23 = 5^{(e_1+e_2 \ mod \ 23)}$$
$$\text{por lo tanto Decodificación}(e_1 + e_2 \text{ mod } 23) = (5^n)(5^m)$$

El otro caso ocurre cuando al menos uno de los estados es Z ($e = Z$) eso implica que $(5^e)(5^m) = (0)(5^m) = 0$ por lo tanto decodificación($Z + e$ mod 23) = decodificación(Z)=0. Considerando ambos casos queda bien definida la regla local de evolución y por lo tanto el autómata celular, ya que una suma de estados módulo 23 corresponde aun producto de $\mathbb{Z}_{23}$.

Autómata celular multiplicador en $\mathbb{Z}_{23}$

En base al análisis anterior ya podemos definir un AC multiplicador $\mathbb{Z}_{23}$ como sigue, consideremos una configuración de vecindad de radio un medio por simplicidad, con un lattice unidimensional, con conjunto estados correspondientes a los de la tabla 10, y con función de transición dada por la siguiente expresión.

$$\delta(x_i, x_d) = \begin{cases} x_c = x_i + x_d \; mod \; 22 & si \quad x_i \neq Z \; y \; x_d \neq Z \\ x_c = Z & si \quad x_i = Z \; o \; x_d = Z \end{cases}$$

donde x_i representa la célula izquierda, x_d la derecha y x_c la célula central en la próxima transición.

Ejemplo de cálculo: consideremos la configuración (8,0,19,3,21) de elementos de $\mathbb{Z}_{23}$ con frontera periódica, el cálculo en el AC se realiza como sigue:

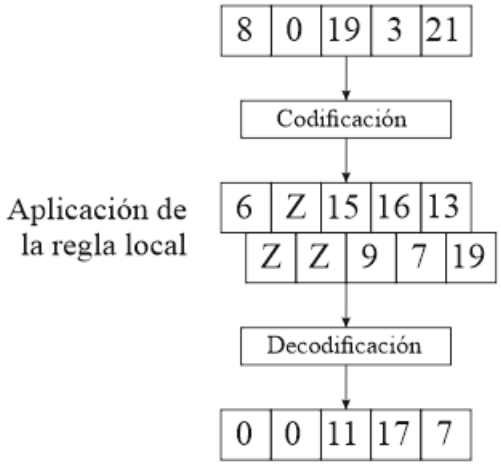

Figura 39: Ejemplo de cálculo.

Realizando las operaciones de multiplicación módulo 23 de manera tradicional tomando dos a dos los elementos de la configuración de ejemplo, obtenemos los siguientes resultados, que corresponden a la decodificación del cálculo realizado por el AC:

- $(8)(0) = 0 \equiv 0 \; mod \; 23$
- $(0)(19) = 0 \equiv 0 \; mod \; 23$
- $(19)(3) = 57 \equiv 11 \; mod \; 23$
- $(3)(21) = 63 \equiv 17 \; mod \; 23$
- $(21)(8) = 168 \equiv 7 \; mod \; 23$

Construcción del AC multiplicador en GF(2^8)

La regla que permite a un autómata celular realizar multiplicaciones de polinomios requiere un análisis matemático más complejo puesto que el producto en GF(2^n) involucra una serie de operaciones que muy difícilmente pueden expresarse en términos de transiciones, ya que en el producto existen acarreos entre los resultados intermedios del producto de polinomios que muy difícilmente presentan patrones, por lo que la idea sobre la cual se fundamenta el

siguiente análisis para la obtención de la regla, es la expresar la regla del producto como una función de transición totalística análogamente a como se hizo en $\mathbb{Z}_{23}$.

Sabemos que si $p(x), q(x) \in$ GF(2^8) entonces su producto, el polinomio $pq(x)$ debe estar en GF(2^8)$_{/m(x)}$ (por cerradura multiplicativa) y como esto se cumple para todos los polinomios en el cuerpo, podemos expresar cualquier polinomio como un producto módulo $m(x)$ de dos ó más polinomios, es decir que si $r(x) \in GF(2^8)$ entonces $r(x) = p_1(x)p_2(x)...p_n(x)$ mod $m(x)$, donde $m(x)$ es el polinomio reductor y $p_i \in$ GF(2^8), pero sabemos de GF(2^8) que por ser campo finito el grupo multiplicativo es asociativo, con lo que solo bastaría encontrar dos polinomios que produzcan a cualquier otro polinomio del cuerpo. Consideremos un AC celular de radio un medio de modo que cada celda contiene un elemento de GF(2^8) tal que si dados dos celulas adyacentes con valores $q(x)$ y $p(x)$ en la siguiente transición el AC debe sustituir la célula central con el producto de producto de las células en el instante anterior, la imagen 40 ilustra la idea de un AC que realiza multiplicaciones en un campo de Galois.

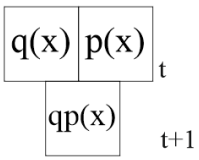

Figura 40: Concepto de un AC multiplicador en campo finito

El problema principal de encontrar la función de transición para la multiplicación es la cantidad de posibles combinaciones de productos ya que como $p(x)$ y $q(x)$ son elementos de GF(2^8) entonces tanto $p(x)$ como $q(x)$ pueden tomar hasta 256 valores posibles cada uno, obteniendo un total de 2^{16} combinaciones, por que la estrategia consiste en extender el análisis realizado para $\mathbb{Z}_{23}$ a los polinomios de GF(2^8). Para GF(2^8) el primer polinomio irreducible es $g(x) = x+1 = (03)_{hex}$ puesto que $g(x)$ no puede ser expresado en términos de dos polinomios en GF(2^8) de grado menor a 1, la propiedad de ser irreducible es una condición necesaria para que nuestro polinomio pueda ser candidato a ser generador. Ahora al igual que en $\mathbb{Z}_{23}$ calculamos las potencias enteras de $g(x)$ módulo $m(x)$, si se repite alguna potencia entonces $x+1$ no será generador donde $m(x) = x^8 + x^4 + x^3 + x + 1$ es el primer polinomio reductor de grado 8 que especifica el estándar AES para el campo GF(2^8).

Calculando todas los posibles valores de $(x+1)^n$ con n $\in \mathbb{Z}_2 55$ se observan que no hay ningún polinomio repetido. Por simplicidad tanto los exponentes como las potencias de $x+1$ módulo $m(x)$ son expresadas como bytes en hexadecimal, siguiendo la notación introducida en la especificación AES para el caso de los polinomios, ahora construimos una función que a cada polinomio $p(x) \in$ GF(2^8) le asigna un natural n con n $\in \mathbb{Z}_{255}$, si es que $p(x) = (x+1)^n \ mod \ m(x)$, a dicha función le denominaremos **logaritmo discreto para polinomios base** $(x+1)$ **(**Log_{x+1}**)** por su analogía con la función logaritmo real.

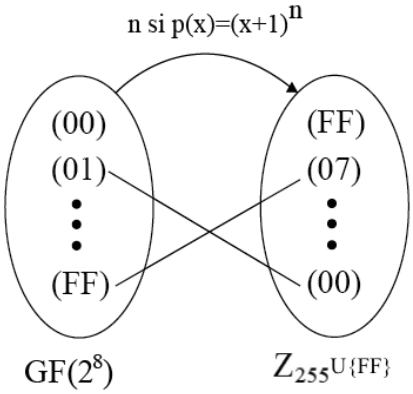

Figura 41: Construcción de la función Logaritmo discreto

Observemos que la función construida no es inyectiva, esto es natural puesto que el campo de Galois GF(2^8), hereda la propiedad de ser dominio de integridad, lo cual implica que no existe ningún natural en la imagen de la función que satisfaga la regla de correspondencia para la instancia (00), de igual manera si consideramos $\mathbb{Z}_{255} \cup \{(0xFF)_{hex}\}$, la función no es suprayectiva puesto que la imagen (FF) bajo la función es congruente con la imagen (00) en el rango de la función, y como se requiere que la función sea univaluada no la asignamos ningún valor en el dominio. Sin embargo es necesario cubrir todas las instancias de información posibles incluido el byte con valor (0x00) en el dominio y (0xFF) en el rango, por lo que nos vemos obligados a agregar la siguiente regla de correspondencia $(x+1)^{FF} = (00)$. Una observación importante es que matemáticamente esa operación no es correcta, ya que la potencia (0xFF) pues es congruente con la potencia 0 en $\mathbb{Z}_{255}$, tal que $(x+1)^0 \equiv (x+1)^{FF}$, de modo que $(x+1)^{FF} = (0,0)_{hex}$. A esta función construida a partir del logaritmo discreto, la denominaremos **caja logarítmica abreviada como: L-Box** de modo que el rango de la función sera $\mathbb{Z}_{255} \cup \{(0xFF)_{hex}\}$.

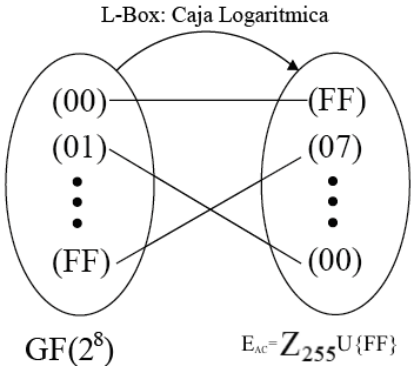

Figura 42: Construcción de la caja logarítmica: L-Box

La construcción anterior admite una representación gráfica, dada por un grafo tipo ciclo, es por ello que se dice que el grupo multiplicativo bajo el generador $g(x) = (x+1)$, es un grupo cílico, la gráfica del grupo se muestra a continuación.

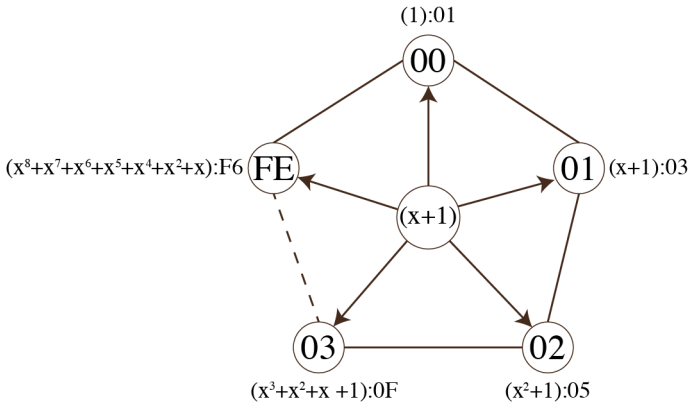

Figura 43: Grafó asociado al grupo cílico multiplicativo de GF(2^8)

Como se observa en la gráfica, el vértice central es el elemento generador, los vértices del ciclo son las potencias del generador, cada vértice esta etiquetado con su correspondiente polinomio generado seguido de su numeración canónica, es fácil ver que la gráfica induce una nueva numeración determinada por las longitudes de las secuencias de $(x+1)$ necesarias para generar al polinomio etiquetado, esto es natural pues dichas longitudes son nuestros estados, habiendo un total de 255 vértices.

Siguiendo la misma mecánica que en la tabla 10, se construye la tabla 11 donde las columnas con encabezado P son las potencias de $(x+1)$ modulo $m(x)$ y la columnas con encabezado E representa el estado (ó exponente), de igual manera la tabla obtenida es colectivamente exhaustiva y con hasta 255 posibles valores para los exponentes, dicha tabla corresponde a la función L-Box.

P	E	P	E	P	E	P	E	P	E	P	E	P	E	P	E
0	FF	**1**	0	**2**	19	**3**	1	**4**	32	**5**	2	**6**	1A	**7**	C6
8	4B	**9**	C7	**A**	1B	**B**	68	**C**	33	**D**	EE	**E**	DF	**F**	3
10	64	**11**	4	**12**	E0	**13**	E	**14**	34	**15**	8D	**16**	81	**17**	EF
18	4C	**19**	71	**1A**	8	**1B**	C8	**1C**	F8	**1D**	69	**1E**	1C	**1F**	C1
20	7D	**21**	C2	**22**	1D	**23**	B5	**24**	F9	**25**	B9	**26**	27	**27**	6A
28	4D	**29**	E4	**2A**	A6	**2B**	72	**2C**	9A	**2D**	C9	**2E**	9	**2F**	78
30	65	**31**	2F	**32**	8A	**33**	5	**34**	21	**35**	F	**36**	E1	**37**	24
38	12	**39**	F0	**3A**	82	**3B**	45	**3C**	35	**3D**	93	**3E**	DA	**3F**	8E
40	96	**41**	8F	**42**	DB	**43**	BD	**44**	36	**45**	D0	**46**	CE	**47**	94
48	13	**49**	5C	**4A**	D2	**4B**	F1	**4C**	40	**4D**	46	**4E**	83	**4F**	38
50	66	**51**	DD	**52**	FD	**53**	30	**54**	BF	**55**	6	**56**	8B	**57**	62
58	B3	**59**	25	**5A**	E2	**5B**	98	**5C**	22	**5D**	88	**5E**	91	**5F**	10
60	7E	**61**	6E	**62**	48	**63**	C3	**64**	A3	**65**	B6	**66**	1E	**67**	42
68	3A	**69**	6B	**6A**	28	**6B**	54	**6C**	FA	**6D**	85	**6E**	3D	**6F**	BA
70	2B	**71**	79	**72**	A	**73**	15	**74**	9B	**75**	9F	**76**	5E	**77**	CA
78	4E	**79**	D4	**7A**	AC	**7B**	E5	**7C**	F3	**7D**	73	**7E**	A7	**7F**	57
80	AF	**81**	58	**82**	A8	**83**	50	**84**	F4	**85**	EA	**86**	D6	**87**	74
88	4F	**89**	AE	**8A**	E9	**8B**	D5	**8C**	E7	**8D**	E6	**8E**	AD	**8F**	E8
90	2C	**91**	D7	**92**	75	**93**	7A	**94**	EB	**95**	16	**96**	B	**97**	F5
98	59	**99**	CB	**9A**	5F	**9B**	B0	**9C**	9C	**9D**	A9	**9E**	51	**9F**	A0
A0	7F	**A1**	C	**A2**	F6	**A3**	6F	**A4**	17	**A5**	C4	**A6**	49	**A7**	EC
A8	D8	**A9**	43	**AA**	1F	**AB**	2D	**AC**	A4	**AD**	76	**AE**	7B	**AF**	B7
B0	CC	**B1**	BB	**B2**	3E	**B3**	5A	**B4**	FB	**B5**	60	**B6**	B1	**B7**	86
B8	3B	**B9**	52	**BA**	A1	**BB**	6C	**BC**	AA	**BD**	55	**BE**	29	**BF**	9D
C0	97	**C1**	B2	**C2**	87	**C3**	90	**C4**	61	**C5**	BE	**C6**	DC	**C7**	FC
C8	BC	**C9**	95	**CA**	CF	**CB**	CD	**CC**	37	**CD**	3F	**CE**	5B	**CF**	D1
D0	53	**D1**	39	**D2**	84	**D3**	3C	**D4**	41	**D5**	A2	**D6**	6D	**D7**	47
D8	14	**D9**	2A	**DA**	9E	**DB**	5D	**DC**	56	**DD**	F2	**DE**	D3	**DF**	AB
E0	44	**E1**	11	**E2**	92	**E3**	D9	**E4**	23	**E5**	20	**E6**	2E	**E7**	89
E8	B4	**E9**	7C	**EA**	B8	**EB**	26	**EC**	77	**ED**	99	**EE**	E3	**EF**	A5
F0	67	**F1**	4A	**F2**	ED	**F3**	DE	**F4**	C5	**F5**	31	**F6**	FE	**F7**	18
F8	D	**F9**	63	**FA**	8C	**FB**	80	**FC**	C0	**FD**	F7	**FE**	70	**FF**	7

Cuadro 11: Función: Caja logarítmica base $x+1$ módulo $x^8+x^4+x^3+x+1$.

La tabla anterior sirve para codificar la información a estados que el AC pueda calcular para realizar multiplicaciones de polinomios en GF(2^8), al igual que la tabla 10 esta induce una función entre los elementos de $\mathbb{Z}_{255} \cup \{(0xFF)_{hex}\}$ y los de GF(2^8), además podemos asegurar dicha función es biyectiva por que cumple las siguientes condiciones.

- No existe un valor en la tabla tal que este le correspondan dos estados diferentes.
- Para cualesquiera dos estados diferentes en la tabla implican valores diferentes.

Por ser biyectiva la función implica que tiene inversa, a la función inversa la denominaremos caja antilogarítmica (abreviada A-Box) de igual manera por su analogía con la función antilogaritmo (ó exponencial) real. Como la tabla no es manejable en caso de que se quiera encontrar el antilogarítmo de un estado se crea la tabla 12 que es similar a la tabla 11 solo que ordenada por estado.

E	P	E	P	E	P	E	P	E	P	E	P	E	P	E	P
FF	0	**0**	1	**1**	3	**2**	5	**3**	F	**4**	11	**5**	33	**6**	55
7	FF	**8**	1A	**9**	2E	**A**	72	**B**	96	**C**	A1	**D**	F8	**E**	13
F	35	**10**	5F	**11**	E1	**12**	38	**13**	48	**14**	D8	**15**	73	**16**	95
17	A4	**18**	F7	**19**	2	**1A**	6	**1B**	A	**1C**	1E	**1D**	22	**1E**	66
1F	AA	**20**	E5	**21**	34	**22**	5C	**23**	E4	**24**	37	**25**	59	**26**	EB
27	26	**28**	6A	**29**	BE	**2A**	D9	**2B**	70	**2C**	90	**2D**	AB	**2E**	E6
2F	31	**30**	53	**31**	F5	**32**	4	**33**	C	**34**	14	**35**	3C	**36**	44
37	CC	**38**	4F	**39**	D1	**3A**	68	**3B**	B8	**3C**	D3	**3D**	6E	**3E**	B2
3F	CD	**40**	4C	**41**	D4	**42**	67	**43**	A9	**44**	E0	**45**	3B	**46**	4D
47	D7	**48**	62	**49**	A6	**4A**	F1	**4B**	8	**4C**	18	**4D**	28	**4E**	78
4F	88	**50**	83	**51**	9E	**52**	B9	**53**	D0	**54**	6B	**55**	BD	**56**	DC
57	7F	**58**	81	**59**	98	**5A**	B3	**5B**	CE	**5C**	49	**5D**	DB	**5E**	76
5F	9A	**60**	B5	**61**	C4	**62**	57	**63**	F9	**64**	10	**65**	30	**66**	50
67	F0	**68**	B	**69**	1D	**6A**	27	**6B**	69	**6C**	BB	**6D**	D6	**6E**	61
6F	A3	**70**	FE	**71**	19	**72**	2B	**73**	7D	**74**	87	**75**	92	**76**	AD
77	EC	**78**	2F	**79**	71	**7A**	93	**7B**	AE	**7C**	E9	**7D**	20	**7E**	60
7F	A0	**80**	FB	**81**	16	**82**	3A	**83**	4E	**84**	D2	**85**	6D	**86**	B7
87	C2	**88**	5D	**89**	E7	**8A**	32	**8B**	56	**8C**	FA	**8D**	15	**8E**	3F
8F	41	**90**	C3	**91**	5E	**92**	E2	**93**	3D	**94**	47	**95**	C9	**96**	40
97	C0	**98**	5B	**99**	ED	**9A**	2C	**9B**	74	**9C**	9C	**9D**	BF	**9E**	DA
9F	75	**A0**	9F	**A1**	BA	**A2**	D5	**A3**	64	**A4**	AC	**A5**	EF	**A6**	2A
A7	7E	**A8**	82	**A9**	9D	**AA**	BC	**AB**	DF	**AC**	7A	**AD**	8E	**AE**	89
AF	80	**B0**	9B	**B1**	B6	**B2**	C1	**B3**	58	**B4**	E8	**B5**	23	**B6**	65
B7	AF	**B8**	EA	**B9**	25	**BA**	6F	**BB**	B1	**BC**	C8	**BD**	43	**BE**	C5
BF	54	**C0**	FC	**C1**	1F	**C2**	21	**C3**	63	**C4**	A5	**C5**	F4	**C6**	7
C7	9	**C8**	1B	**C9**	2D	**CA**	77	**CB**	99	**CC**	B0	**CD**	CB	**CE**	46
CF	CA	**D0**	45	**D1**	CF	**D2**	4A	**D3**	DE	**D4**	79	**D5**	8B	**D6**	86
D7	91	**D8**	A8	**D9**	E3	**DA**	3E	**DB**	42	**DC**	C6	**DD**	51	**DE**	F3
DF	E	**E0**	12	**E1**	36	**E2**	5A	**E3**	EE	**E4**	29	**E5**	7B	**E6**	8D
E7	8C	**E8**	8F	**E9**	8A	**EA**	85	**EB**	94	**EC**	A7	**ED**	F2	**EE**	D
EF	17	**F0**	39	**F1**	4B	**F2**	DD	**F3**	7C	**F4**	84	**F5**	97	**F6**	A2
F7	FD	**F8**	1C	**F9**	24	**FA**	6C	**FB**	B4	**FC**	C7	**FD**	52	**FE**	F6

Cuadro 12: Tabla de decodificación de estados base $x+1$ módulo $x^8+x^4+x^3+x+1$.

Una ventaja de utilizar este esquema basado en logaritmos discretos es que con un byte podemos representar todos los polinomios de GF(2^8) así como lo de sus logaritmos discretos $\mathbb{Z}_{255} \cup \{(0xFF)_{hex}\}$ i.e. los naturales entre 0 y 254 unión 255, para simplificar, llamamos al conjunto de todas las imágenes de L-Box ($\mathbb{Z}_{255}$) llamaremos **Espacio de estados:** E_{AC}. Ahora con el objetivo de facilitar la búsqueda en las tablas, se adopta la estrategia usada en la construcción de las cajas S de AES, de modo que se reestructuran las tablas anteriores 11 y 12 de tal forma que en la primera tabla 13, la parte de alta de P corresponde al número de renglón y la parte baja de P corresponde al número de columna; para la segunda tabla 12, la parte de alta de E corresponde al número de renglón y la parte baja de E corresponde al número de columna.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	**00**	19	**01**	32	**02**	1A	**C6**	4B	**C7**	1B	**68**	33	**EE**	DF	**03**
1	64	**04**	E0	**0E**	34	**8D**	81	**EF**	4C	**71**	08	**C8**	F8	**69**	1C	**C1**
2	7D	**C2**	1D	**B5**	F9	**B9**	27	**6A**	4D	**E4**	A6	**72**	9A	**C9**	09	**78**
3	65	**2F**	8A	**05**	21	**0F**	E1	**24**	12	**F0**	82	**45**	35	**93**	DA	**8E**
4	96	**8F**	DB	**BD**	36	**D0**	CE	**94**	13	**5C**	D2	**F1**	40	**46**	83	**38**
5	66	**DD**	FD	**30**	BF	**06**	8B	**62**	B3	**25**	E2	**98**	22	**88**	91	**10**
6	7E	**6E**	48	**C3**	A3	**B6**	1E	**42**	3A	**6B**	28	**54**	FA	**85**	3D	**BA**
7	2B	**79**	0A	**15**	9B	**9F**	5E	**CA**	4E	**D4**	AC	**E5**	F3	**73**	A7	**57**
8	AF	**58**	A8	**50**	F4	**EA**	D6	**74**	4F	**AE**	E9	**D5**	E7	**E6**	AD	**E8**
9	2C	**D7**	75	**7A**	EB	**16**	0B	**F5**	59	**CB**	5F	**B0**	9C	**A9**	51	**A0**
A	7F	**0C**	F6	**6F**	17	**C4**	49	**EC**	D8	**43**	1F	**2D**	A4	**76**	7B	**B7**
B	CC	**BB**	3E	**5A**	FB	**60**	B1	**86**	3B	**52**	A1	**6C**	AA	**55**	29	**9D**
C	97	**B2**	87	**90**	61	**BE**	DC	**FC**	BC	**95**	CF	**CD**	37	**3F**	5B	**D1**
D	53	**39**	84	**3C**	41	**A2**	6D	**47**	14	**2A**	9E	**5D**	56	**F2**	D3	**AB**
E	44	**11**	92	**D9**	23	**20**	2E	**89**	B4	**7C**	B8	**26**	77	**99**	E3	**A5**
F	67	**4A**	ED	**DE**	C5	**31**	FE	**18**	0D	**63**	8C	**80**	C0	**F7**	70	**07**

Cuadro 13: L-Box: caja logarítmica

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	01	**03**	05	**0F**	11	**33**	55	**FF**	1A	**2E**	72	**96**	A1	**F8**	13	**35**
1	5F	**E1**	38	**48**	D8	**73**	95	**A4**	F7	**02**	06	**0A**	1E	**22**	66	**AA**
2	E5	**34**	5C	**E4**	37	**59**	EB	**26**	6A	**BE**	D9	**70**	90	**AB**	E6	**31**
3	53	**F5**	04	**0C**	14	**3C**	44	**CC**	4F	**D1**	68	**B8**	D3	**6E**	B2	**CD**
4	4C	**D4**	67	**A9**	E0	**3B**	4D	**D7**	62	**A6**	F1	**08**	18	**28**	78	**88**
5	83	**9E**	B9	**D0**	6B	**BD**	DC	**7F**	81	**98**	B3	**CE**	49	**DB**	76	**9A**
6	B5	**C4**	57	**F9**	10	**30**	50	**F0**	0B	**1D**	27	**69**	BB	**D6**	61	**A3**
7	FE	**19**	2B	**7D**	87	**92**	AD	**EC**	2F	**71**	93	**AE**	E9	**20**	60	**A0**
8	FB	**16**	3A	**4E**	D2	**6D**	B7	**C2**	5D	**E7**	32	**56**	FA	**15**	3F	**41**
9	C3	**5E**	E2	**3D**	47	**C9**	40	**C0**	5B	**ED**	2C	**74**	9C	**BF**	DA	**75**
A	9F	**BA**	D5	**64**	AC	**EF**	2A	**7E**	82	**9D**	BC	**DF**	7A	**8E**	89	**80**
B	9B	**B6**	C1	**58**	E8	**23**	65	**AF**	EA	**25**	6F	**21**	C8	**43**	C5	**54**
C	FC	**1F**	21	**63**	A5	**F4**	07	**09**	1B	**2D**	77	**99**	B0	**CB**	46	**CA**
D	45	**CF**	4A	**DE**	79	**8B**	86	**91**	A8	**E3**	3E	**42**	C6	**51**	F3	**0E**
E	12	**36**	5A	**EE**	29	**7B**	8D	**8C**	8F	**8A**	85	**94**	A7	**F2**	0D	**17**
F	39	**4B**	DD	**7C**	84	**97**	A2	**FD**	1C	**24**	6C	**B4**	C7	**52**	F6	**00**

Cuadro 14: A-Box: caja antilogarítmica

Caracterización 1: sobre el del producto en GF(2^8) y el espacio de estados
Sean las imágenes e_1 y e_2 elementos del espacio de estados E_{AC}, bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces e_1 + $e_2 \in Z_{255}$ si y solo si $p_1(x) \bullet p_2(x) \neq 0 \in$ GF(2^8)

Demostración:
De la construcción anterior se puede determinar que $1 = (x+1)^0$ mod $m(x)$, como 1 es el neutro bajo la operacion $\bullet$ este determina el orden del grupo multiplicativo, por lo que en general $1 = (x+1)^{255k}$ donde $k \in \mathbb{Z}$. Por otro lado si e_1 y e_2 son elementos de E_{AC}, entonces son estados de la tabla 13, por hipótesis como ambos estados $e_1 \neq FF$ y $e_2 \neq FF$, implica que

existen $e_1 = n$ y $e_2 = m$ para algún n y $m \in \{1, ..., 255\}$ tales que $p_1(x) = g(x)^n$, $p_2(x) = g(x)^m$ $\in$ GF(2^8)$_{/m(x)}$ donde $g(x) = (x + 1)$,

$$\begin{aligned} p_1(x) = g(x)^n = g(x)^n \bullet (1) &= g(x)^n \bullet g(x)^{255k_1} = g(x)^{n+255k_1} \\ L_{BOX}[p_1(x)] &= L_{BOX}[g(x)^{n+255k_1}] \\ e_1 &= n + 255k_1 \\ e_1 &= n \mod 255 \\ p_2(x) = g(x)^m = g(x)^m \bullet (1) &= g(x)^m \bullet g(x)^{255k_2} = g(x)^{m+255k_2} \\ L_{BOX}[p_2(x)] &= L_{BOX}[g(x)^{m+255k_2}] \\ e_2 &= m + 255k_2 \\ e_2 &= m \mod 255 \end{aligned}$$

Con k_1 y $k_2 \in \mathbb{Z}$, desarollando el producto tenemos que:

$$\begin{aligned} p_1(x) \bullet p_2(x) \mod m(x) &= g(x)^{n+255k_1} \bullet g(x)^{m+255k_2} \mod m(x) \\ p_1(x) \bullet p_2(x) \mod m(x) &= g(x)^{n+m+255k_1+255k_2} \mod m(x) \\ L_{BOX}[p_1(x) \bullet p_2(x)] &= L_{BOX}[g(x)^{(n+m)+255(k_1+k_2)}] \\ L_{BOX}[p_1(x) \bullet p_2(x)] &= (n+m) + 255(k_1+k_2) \quad como \quad (n+m),(k_1+k_2) \in \mathbb{Z} \\ L_{BOX}[p_1(x) \bullet p_2(x)] &= e_1 + e_2 \mod 255 \end{aligned}$$

La proposición anterior muestra que la L-Box tiene la propiedad de abrir el la imagen del producto de 2 polinomios en la suma de sus imágenes correspondientes. En otras palabras, que sumar estados adyacentes en E_{AC} equivale a multiplicar polinomios en GF(2^8).

Caracterización 2: sobre el dominio de integridad de GF(2^8) y el espacio de estados
Sean las imágenes de e_1 y e_2 elementos del espacio de estados E_{AC}, bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces $p_1(x) \bullet p_2(x) = 0 \in$ GF(2^8) si y solo si $Max\{e_1, e_2\} = 0xFF$.

Demostración:
Como de e_1 y e_2 no sabemos nada, si no únicamente que por hipótesis $p_1(x) \bullet p_2(x) = 0$.

$$\begin{aligned} p_1(x)p_2(x) &= 0 \quad por \quad ser \quad GF(2^8) \quad dominio \quad de \quad integridad \\ L_{BOX}[p_1(x)p_2(x)] &= L_{BOX}[0] \quad \in \quad E_{AC} \\ L_{BOX}[p_1(x)p_2(x)] &= FF \quad \notin \quad Z_{255} \quad por \quad lo \quad que \quad L_{BOX} \quad no \quad abre \quad el \quad producto \\ & \quad pero \quad 0xFF \quad es \quad el \quad valor \quad mas \quad alto \quad que \quad puede \quad tomar \quad L_{BOX} \\ L_{BOX}[p_1(x)] = FF \quad \vee & \quad L_{BOX}[p_1(x)] = FF \\ e_1 = FF \quad \vee & \quad e_2 = FF \quad pues \quad L_{BOX}[p_1(x)p_2(x)] = FF \\ L_{BOX}[p_1(x)p_2(x)] &= max\{e_1, e_2\} = FF \end{aligned}$$

La proposición anterior muestra que L-Box debe respetar el dominio de integridad del campo, por lo que podemos deducir que la caracterización del producto en términos de estados genera un imagen de L-Box de 0 (a saber FF) si al menos uno de los dos estados que se reciben son FF.

Caracterización 3: sobre la conmutatividad del producto en GF(2^8) y el espacio de estados
Si e_1 y e_2 son imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, $p_1(x) \bullet p_2(x) =$

$p_2(x) \bullet p_1(x) \in$ GF(2^8) si y solo si si y solo si $L_{Box}[p_1(x) \bullet p_2(x)]$=$L_{Box}[p_2(x) \bullet p_1(x)]$.

Demostración:
Directa de las demostraciones de caracterización 1 y 2. Pues en la caracterización 1: $p_1(x) \bullet p_2(x) = p_2(x) \bullet p_1(x) \Rightarrow L_{BOX}[p_1(x) \bullet p_2(x)] = L_{BOX}[p_2(x) \bullet p_1(x)] \Rightarrow e_1 + e_2 = e_2 + e_1 \in Z_{255}$ y en la caracterización 2: $max\{e_1, e_2\} = max\{e_2, e_1\} = FF$.

Caracterización 4: sobre los inversos multiplicativos de GF(2^8) y el espacio de estados
Sean $e_1 \neq FF \neq e_2$ imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces $p_1(x) \bullet p_2(x) = 1 \mod m(x) \in$ GF(2^8) si y solo si $\frac{1}{p_1(x)} = A_{box}[255 - e_1]$.

Demostración:
por hipótesis como $e_1 \neq FF \neq e_2 \Rightarrow p_1(x) \neq 0 \neq p_2(x)$ y además sabemos que:

$$\begin{aligned} p_1(x) \bullet p_2(x) &= 1 \mod m(x) \\ p_2(x) \bullet p_1(x) &= 1 \mod m(x) \\ p_2(x) &= \frac{1}{p_1(x)} \mod m(x) \quad Ec.\ 1 \end{aligned}$$

Esto es por un lado, por otro como $e_1 \neq FF \neq e_2$ implica que existen $e_1 = n$ y $e_2 = m$ para algún n y m $\in Z_{255}$ tales que $g(x)^n$, $g(x)^m \in$ GF(2^8)$_{/m(x)}$ donde $g(x) = (x + 1)$.

$$\begin{aligned} g(x)^n g(x)^m &= 1 \mod m(x) \\ L_{BOX}[g(x)^n g(x)^m] &= L_{BOX}[1] \in Z_{255} \\ L_{BOX}[g(x)^n] + L_{BOX}[g(x)^m] &= 0 \mod 255 \ aplicamos\ C-1 \\ e_1 + e_2 &= 0 \mod 255 \\ e_2 &= -e_1 \mod 255 \ por\ propiedad\ de\ congruencias \\ A_{BOX}[e_2] &= A_{BOX}[(-e_1) \mod 255] \in GF(2^8) \\ p_2(x) &= A_{BOX}[255 - e_1] \ Sustituyendo\ Ec.\ 1\ tenemos \\ \frac{1}{p_1(x)} &= A_{BOX}[255 - e_1] \end{aligned}$$

El presente resultado es notable, pues permite calcular los inversos multiplicativos del campo GF(2^8) de una manera muy sencilla utilizando los estados del autómata mediante la siguiente formula, sin necesidad de aplicar algoritmo de euclides extendido para polinomios.

$$\forall \quad p_1(x) \neq 0 \quad \in \quad GF(2^8) \quad \frac{1}{p_1(x)} = A_{BOX}[255 - L_{BOX}[p_1(x)]]$$

Autómata celular multiplicador en GF(2^8)
Ahora ya obtenidas las tablas logarítmica y antilogarítmica, y aplicando las caracterizaciones 1, 2 y 3 para determinar una función de transición, es posible definir un AC multiplicador en GF(2^8) en términos del espacio de estados E_{AC} determinado por L-Box como sigue: consideremos una configuración de vecindad de radio un medio por simplicidad, con un lattice unidimensional, con conjunto estados correspondientes a los de la tabla 13, y con función de transición dada por la siguiente expresión.

$$\boxdot(x_{c_t}, x_{d_t}) = \begin{cases} x_{c_{t+1}} = x_{c_t} + x_{d_t} \ mod\ 255 & si \quad x_{c_t} \neq FF\ y\ x_{d_t} \neq FF \\ x_{c_{t+1}} = 00 & si \quad x_{c_t} = FF\ o\ x_{d_t} = FF \end{cases}$$

Donde x_{c_t} representa el valor la célula central en el instante t, x_{d_t} el valor de la célula adyacente a la derecha en el instante t y $x_{c_{t+1}}$ el valor de la célula central en el instante t+1.

Ejemplo de cálculo: consideremos la configuración (52,F6,03,BF,D4) de elementos de GF(2^8) con frontera periódica, el cálculo en el AC se realiza como sigue:

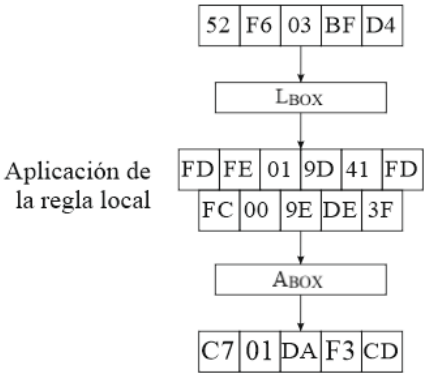

Figura 44: Ejemplo de cálculo de un AC multiplicador en GF(2^8).

Realizando las operaciones de multiplicación de polinomios de manera tomando dos a dos los elementos de la configuración de ejemplo, obtenemos los siguientes resultados, que corresponden a la decodificación del cálculo realizado por el AC:

- $(52)(F6) = (x^6+x^4+x)(x^7+x^6+x^5+x^4+x^2+x) \equiv x^7+x^6+x^2+x+1 \; mod \; m(x) = (C7)$
- $(F6)(03) = (x^7+x^6+x^5+x^4+x^2+x)(x+1) \equiv 1 \; mod \; m(x) = (01)$
- $(03)(BF) = (x+1)(x^7+x^5+x^4+x^3+x^2+x+1) \equiv x^7+x^6+x^4+x^3+x+1 \; mod \; m(x)$
- $(BF)(D4) = (x^7+x^5+x^4+x^3+x^2+x+1)(x^7+x^6+x^4+x^2) \equiv x^7+x^6+x^5+x^4+x+1 \; mod \; m(x) = (F3)$
- $(D4)(52) = (x^7+x^6+x^4+x^2)(x^6+x^4+x) \equiv x^5+x^4+x^3+x^2+x+1 \; mod \; m(x) = (CD)$

Teniendo como precedente el modelado matemático anteriormente desarrollado y una vez determinada la función de transición del autómata celular multiplicador en GF 2^8, es momento de implementar la multiplicación polinomial. En la presente sección se muestra con algunos gráficos, la implementación de la multiplicación polinomial del sistema.

Las siguientes gráficas muestran que el autómata celular, presentan ciertos comportamientos que se asemejan a la clasificación de Wolfram, con algunas entradas especificas:

A continuación se describen las entradas del sistema para cada uno de los casos.

- **Introduce la configuración inicial:** se introduce una lista de valores decimales no mayores a 255, tomando a esta lista como la configuración inicial del autómata.
- **Número de evoluciones:** se introduce un número decimal, tomándolo como el número de veces en que el autómata habrá de recalcularse.
- **Tamaño de célula:** se introduce un número decimal que habrá de tomarse como el tamaño del nibble en pixeles.
- **Configuración:** se introduce un número decimal que ha de tomarse como la evolución elegida que ha de mostrarse en el cuadro diálogo.

Prueba: Clase I
Evolución a un estado uniforme. Después de transcurrido un cierto número de generaciones, todas las células del autómata convergen a un solo estado.

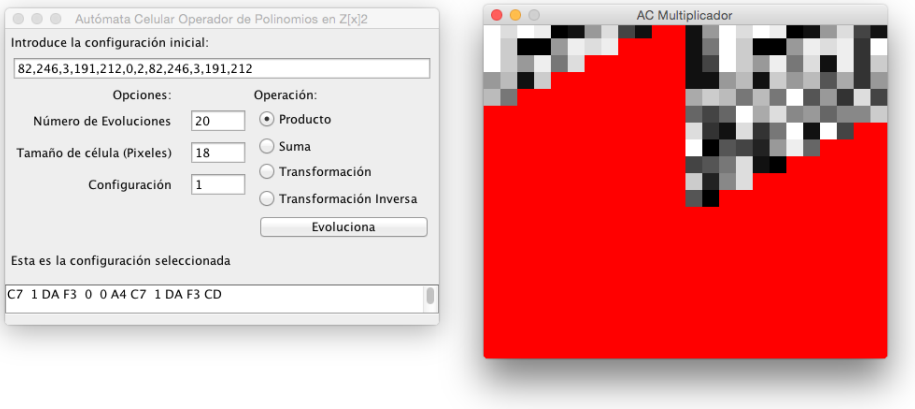

Figura 45: Ejemplo de cálculo clase I, el estado FF se propaga por la lattice siempre.

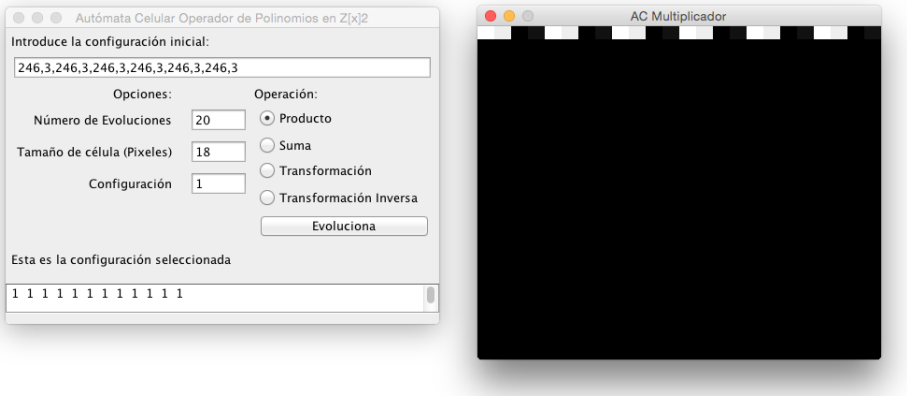

Figura 46: Ejemplo de cálculo clase I, el estado 0 se propaga por la lattice cuando todos son inversos multiplicativos.

Prueba: Clase II

La evolución del autómata celular lleva a un conjunto de estructuras simples que son estables o periódicas.

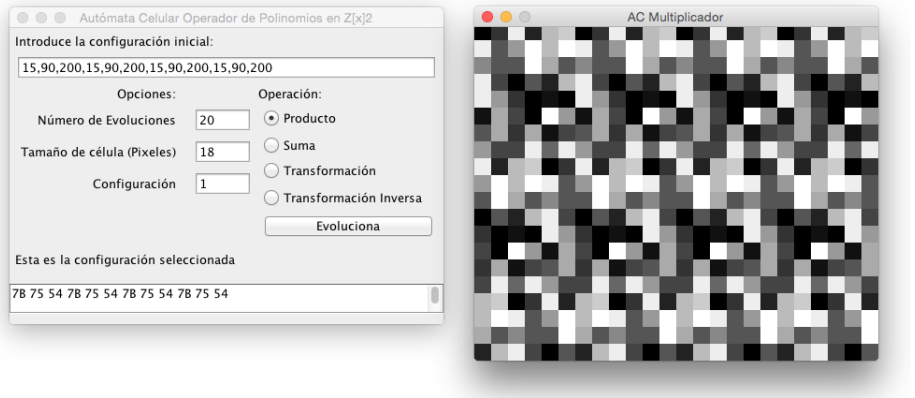

Figura 47: Ejemplo de cálculo clase II para el autómata multiplicador.

La aparición de patrones periodicos es un comportamiento bastante interesante, pues puede indicar la posible concepción de una sub estructura algebraica conocida como subgrupo multiplicativo cíclico, pues el autómata multiplicador entra en un ciclo periodico de manera indeterminada bajo ciertas entradas, esto podría deberse a que los elementos de la primera

configuración son elementos de tal subgrupo. La detección de dichas sub estructuras algebraicas por lo general no es trivial ya que además de no ser comunes, su formación depende de la operación sobre la cual esta definida el grupo en general.

Prueba: Clase III
La evolución del autómata celular lleva a un conjunto siempre caótico.

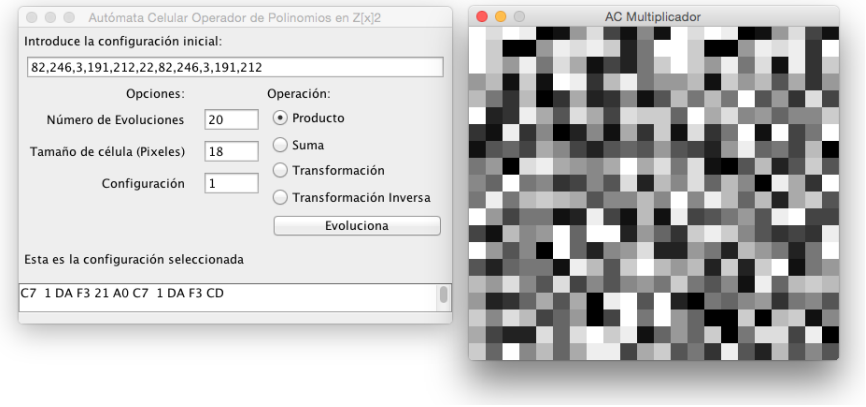

Figura 48: Ejemplo de cálculo clase III para el autómata multiplicador.

5.1.2. Autómata celular transformador

La caracterización del producto en el campo finito GF(2^8) determina totalmente la caracterización de las demás operaciones primitivas, en el sentido que es necesario obtener las reglas para los autómatas celulares, transformador y sumador, de manera que su espacio de estados sea el obtenido por L-Box, ahora bien la siguiente operación en ser caracterizada es la **Transformación lineal affín** denominada **Caja de sustitución (S-Box)**, recordemos que la transformación lineal affin especificada en el estándar es biyectiva y que opera directamente sobre los coeficientes de polinomios en GF(2^8), el objetivo de esta caracterización es obtener una nueva función biyectiva que opere sobre el espacio de estados E_{AC} de modo que las transformaciones que realizan sean equivalentes a su homologa (S-Box) en el campo GF(2^8).

Caracterización 5: sobre la caja S definida en GF(2^8) y el espacio de estados
Sean $e_1 \quad y \quad e_2$ imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces existe una función biyectiva $f : E_{AC} \to E_{AC}$ tal que $S_{BOX}[p_1(x)] = p_2(x) \in$ GF(2^8) $\Leftrightarrow f(e_1) = e_2 \in E_{AC}$.

Demostración:
En primer lugar debemos tener en cuenta el siguiente resultado: sean $f : A \to B$ y $g : B \to C$ funciones biyectivas, la composición $g \circ f$ es biyecta y por lo tanto inversible con inversa a saber $f^{-1} \circ g^{-1}$. Por hipótesis tenemos que $L_{BOX}[p_1(x)] = e_1$ y $L_{BOX}[p_2(x)] = e_2$, establezcamos el siguiente diagrama conmutativo para poder probar la existencia de f:

$$\begin{array}{ccc} E_{AC} & \xrightarrow{\exists f} & E_{AC} \\ \downarrow A_{BOX} & & \uparrow L_{BOX} \\ GF(2^8) & \xrightarrow{S_{Box}} & GF(2^8) \end{array}$$

Consideremos las funciones $A_{BOX} : E_{AC} \to GF(2^8)$ y $S_{BOX} : GF(2^8) \to GF(2^8)$, que figuran en el diagrama, ambas son biyectivas pues A_{BOX} es la inversa de L_{BOX} y S_{BOX} es la transformación affin definida en el estándar de AES. por lo tanto la composición A_{BOX} seguida de S_{BOX} existe y es biyectiva por el resultado anteriormente mencionado, de forma que:

$$S_{BOX} \circ A_{BOX} : E_{AC} \to GF(2^8)$$

Consideremos la tercera función que figura en el diagrama conmutativo, $L_{BOX} : GF(2^8) \to E_{AC}$ que es biyectiva, y realicemos la composición de funciones L_{BOX} seguida de $S_{BOX} \circ A_{BOX}$ que de igual manera existe y es biyectiva.

$$L_{BOX} \circ S_{BOX} \circ A_{BOX} : E_{AC} \to E_{AC}$$

Aseguramos que dicha función es la f que hace conmutar al diagrama anterior y por lo tanto será la función homologa a S_{BOX} en términos de los estados del autómata celular. Para probar esto es necesario partir la demostración en dos implicaciones:

$\Rightarrow$: Dado que existe $f := L_{BOX} \circ S_{BOX} \circ A_{BOX}$ y $S_{BOX}(p_1(x)) = p_2(x) \Rightarrow f(e_1) = e_2$.

$$\begin{aligned}
\Rightarrow f(e_1) &= L_{BOX} \circ S_{BOX} \circ A_{BOX}[e_1] \quad por \quad hipotesis \\
&= L_{BOX} \circ S_{BOX}[A_{BOX}[e_1]] \\
&= L_{BOX} \circ S_{BOX}[p_1(x)] \quad por \quad hipotesis \\
&= L_{BOX}[S_{BOX}[p_1(x)]] \\
&= L_{BOX}[p_2(x)] \quad por \quad hipotesis \\
L_{BOX}[S_{BOX}[A_{BOX}[e_1]]] &= e_2 \\
f(e_1) &= e_2
\end{aligned}$$

$\Leftarrow$: Dado que existe $f := L_{BOX} \circ S_{BOX} \circ A_{BOX}$ y $f(e_1) = e_2 \Rightarrow S_{BOX}(p_1(x)) = p_2(x)$.

$$\begin{aligned}
\Rightarrow f(e_1) &= e_2 \quad por \quad hipotesis \\
L_{BOX}[S_{BOX}[A_{BOX}[e_1]]] &= e_2 \\
A_{BOX}[L_{BOX}[S_{BOX}[A_{BOX}[e_1]]]] &= A_{BOX}[e_2] \\
S_{BOX}[A_{BOX}[e_1]] &= p_2(x) \\
S_{BOX}[p_1(x)] &= p_2(x)
\end{aligned}$$

Por lo tanto la función f existe y es la homologa de la caja de sustitución para GF(2^8), por esta misma analogía denominaremos a la la función f como **caja de sustitución logarítmica, de forma abreviada (LS_{BOX})**. Además esta función es única, la unicidad es directa de aplicar el resultado de composición de funciones biyectivas, es decir que no hay otra función que sea homologa a S_{BOX} en términos de los estados que no sea LS_{BOX}. Notemos que el diagrama ofrece una forma sencilla de calcular la función, aplicando directamente la formula $L_{BOX}[S_{BOX}[A_{BOX}[e_1]]]$ para todo e_1 en la tabla de estados, dicho cálculo no requiere mayor problema y se muestra a continuación.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**F3**	E5	**54**	5E	**A8**	90	**C0**	81	**F6**	2F	**96**	2C	**8A**	8F	**73**	0B
1	**D1**	0D	**C6**	FD	**6E**	E8	**A6**	5C	**3A**	CA	**BA**	42	**0A**	7A	**05**	A4
2	**2A**	4C	**D2**	6B	**5F**	CD	**7C**	18	**19**	7B	**0F**	DD	**7E**	48	**AD**	FC
3	**99**	2E	**ED**	70	**8C**	26	**C8**	F1	**F4**	DA	**D0**	FA	**1E**	A0	**24**	55
4	**E4**	13	**EA**	3C	**11**	92	**D9**	DF	**1F**	F9	**0C**	65	**76**	21	**AA**	61
5	**77**	68	**8B**	2B	**57**	AC	**D6**	89	**33**	CE	**85**	D5	**45**	52	**12**	3B
6	**A2**	F8	**98**	CB	**CF**	32	**30**	E7	**72**	17	**37**	63	**B8**	FE	**A5**	1B
7	**6C**	41	**4A**	07	**EF**	38	**16**	5B	**8D**	6F	**56**	23	**1C**	86	**53**	44
8	**03**	94	**AF**	78	**00**	35	**43**	B9	**40**	EB	**B5**	BB	**C9**	25	**9F**	50
9	**09**	B3	**59**	6A	**7F**	F2	**C7**	A1	**F0**	06	**79**	75	**D3**	4B	**62**	A9
A	**5D**	C5	**01**	BD	**D7**	AB	**20**	DE	**0E**	91	**B6**	51	**9E**	71	**EC**	3F
B	**34**	83	**4E**	28	**B0**	27	**46**	D4	**74**	8E	**D8**	BC	**B4**	08	**49**	7D
C	**CC**	97	**F7**	80	**1A**	9D	**BE**	00	**B7**	14	**31**	E3	**89**	C1	**E2**	9B
D	**3D**	E9	**6D**	69	**B1**	93	**36**	58	**87**	04	**3E**	9A	**FB**	39	**EE**	2D
E	**95**	02	**29**	4D	**C4**	C2	**88**	A3	**15**	A7	**F5**	1D	**22**	AE	**47**	67
F	**E0**	5A	**B2**	64	**10**	4F	**82**	BF	**9C**	E1	**66**	E6	**DC**	FF	**DB**	C3

Cuadro 15: Caja de sustitución logarítmica: LS-Box

Caracterización 6: sobre la inversa de la caja S definida en GF(2^8) y el espacio de estados

Sean $e_1 \quad y \quad e_2$ imágenes bajo L-Box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces existe una función biyectiva $f^{-1} : E_{AC} \rightarrow E_{AC}$ tal que $invS_{BOX}[p_2(x)] = p_1(x) \in$ GF(2^8) $\Leftrightarrow f^{-1}(e_2) = e_1 \in E_{AC}$.

Demostración:

La demostración directa es análoga sin embargo es demasiado larga de modo que retomemos la caracterización 5 y recordemos que la composición de funciones biyectivas $g \circ f$ es inversible, con inversa a saber $f^{-1} \circ g^{-1}$, establezcamos el siguiente diagrama conmutativo para poder probar la existencia de f^{-1}.

$$\begin{array}{ccc} E_{AC} & \xleftarrow{\exists f^{-1}} & E_{AC} \\ \uparrow L_{BOX} & & \downarrow A_{BOX} \\ GF(2^8) & \xleftarrow{invS_{Box}} & GF(2^8) \end{array}$$

Retomando la caracterización 5 sabemos que LS_{BOX} existe, es biyectiva y por lo tanto invertible, y además es única, por lo tanto $(LS_{BOX})^{-1}$ existe, es biyectiva, inversible y única. La estrategia consiste en aplicar el resultado de la inversa para la composición de funciones, tengamos en cuenta que invS-Box=S-Box^{-1} es la caja de sustitución inversa establecida en el estándar AES.

$$\begin{aligned} LS_{BOX} &= L_{BOX} \circ S_{BOX} \circ A_{BOX} \\ (LS_{BOX})^{-1} &= (L_{BOX} \circ S_{BOX} \circ A_{BOX})^{-1} \\ &= (A_{BOX}^{-1}) \circ (L_{BOX} \circ S_{BOX})^{-1} \\ &= L_{BOX} \circ (L_{BOX} \circ S_{BOX})^{-1} \\ &= L_{BOX} \circ (S_{BOX}^{-1} \circ L_{BOX}^{-1}) \\ f^{-1} &:= L_{BOX} \circ {}_{Inv}S_{Box} \circ A_{BOX} \end{aligned}$$

Aseguramos que dicha función es la f^{-1} que hace conmutar al diagrama anterior y por lo tanto será la función homologa a $invS_{BOX}$ en términos de los estados del Autómata Celular. Para probar esto es necesario partir la demostración en dos implicaciones:

$\Rightarrow$: Dado que existe y $invS_{BOX}(p_2(x)) = p_1(x) \Rightarrow f^{-1}(e_2) = e_1$.

$$\begin{aligned} LS_{BOX}[e_1] &= e_2 \\ L_{BOX}[S_{BOX}[A_{BOX}[e_1]]] &= e_2 \\ A_{BOX}[L_{BOX}[S_{BOX}[A_{BOX}[e_1]]] &= A_{BOX}[e_2] \\ S_{BOX}[A_{BOX}[e_1] &= A_{BOX}[e_2] \\ {}_{Inv}S_{Box}[S_{BOX}[A_{BOX}[e_1]] &= {}_{Inv}S_{Box}[A_{BOX}[e_2]] \\ A_{BOX}[e_1] &= {}_{Inv}S_{Box}[A_{BOX}[e_2]] \\ L_{BOX}[A_{BOX}[e_1]] &= L_{BOX}[{}_{Inv}S_{BOX}[A_{BOX}[e_2]]] \\ e_1 &= L_{BOX} \circ {}_{Inv}S_{Box} \circ A_{BOX}[e_2] \\ e_1 &= f^{-1}(e_2) \end{aligned}$$

$\Leftarrow$: Dado que existe f^{-1} y $f^{-1}(e_2) = e_1 \Rightarrow invS_{BOX}(p_2(x)) = p_1(x)$. Retomando la caracterización 5, tenemos:

$$\begin{aligned} f^{-1}(e_2) &= e_1 \\ L_{BOX}[{}_{Inv}S_{BOX}[A_{BOX}[e_2]]] &= e_1 \\ {}_{Inv}S_{Box}[A_{BOX}[e_2]] &= A[e_1] \\ {}_{Inv}S_{Box}[A_{BOX}[e_2]] &= p_1(x) \\ {}_{Inv}S_{Box}[p_2(x)] &= p_1(x) \end{aligned}$$

Por lo tanto la función f^{-1} existe, es la homologa de la caja de sustitución inversa para GF(2^8) definida en el estándar AES, por esta misma analogía denominaremos a la la función f como **Caja de inversa sustitución logarítmica, de forma abreviada (**${}_{inv}LS_{BOX}$**)**. Notemos que el diagrama ofrece una forma sencilla de calcular la función, aplicando directamente la formula $L_{BOX}[{}_{Inv}S_{BOX}[A_{BOX}[e_2]]]$ para todo e_2 en la tabla de estados, dicho cálculo no requiere mayor problema y se muestra a continuación.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	C7	**A2**	E1	**80**	D9	**1E**	99	**73**	BD	**90**	1C	**0F**	4A	**11**	A8	**2A**
1	F4	**44**	5E	**41**	C9	**E8**	76	**69**	27	**28**	C4	**6F**	7C	**EB**	3C	**48**
2	A6	**4D**	EC	**7B**	3E	**8D**	35	**B5**	B3	**E2**	20	**53**	0B	**DF**	31	**09**
3	66	**CA**	65	**58**	B0	**85**	D6	**6A**	75	**DD**	18	**5F**	43	**D0**	DA	**AF**
4	88	**71**	1B	**86**	7F	**5C**	B6	**EE**	2D	**BE**	72	**9D**	21	**E3**	B2	**F5**
5	8F	**AB**	5D	**7E**	02	**3F**	7A	**54**	D7	**92**	F1	**77**	17	**A0**	03	**24**
6	84	**4F**	9E	**6B**	F3	**4B**	FA	**EF**	51	**D3**	93	**23**	70	**D2**	14	**79**
7	33	**AD**	68	**0E**	B8	**9B**	4C	**50**	83	**9A**	1D	**29**	26	**BF**	2C	**94**
8	C3	**07**	F6	**B1**	57	**5A**	7D	**D8**	E6	**CC**	0C	**52**	34	**78**	B9	**0D**
9	05	**A9**	45	**D5**	81	**E0**	0A	**C1**	62	**30**	DB	**CF**	F8	**C5**	AC	**8E**
A	3D	**97**	60	**E7**	1F	**6E**	16	**E9**	04	**9F**	4E	**A5**	55	**2E**	ED	**82**
B	B4	**D4**	F2	**91**	BC	**8A**	AA	**C8**	6C	**87**	1A	**8B**	BB	**A3**	C6	**F7**
C	06	**CD**	E5	**FF**	E4	**A1**	12	**96**	36	**8C**	19	**63**	C0	**25**	59	**64**
D	3A	**10**	22	**9C**	B7	**5B**	56	**A4**	BA	**46**	39	**FE**	FC	**2B**	A7	**47**
E	F0	**F9**	CE	**CB**	40	**01**	FB	**67**	15	**D1**	42	**89**	AE	**32**	DE	**74**
F	98	**37**	95	**00**	38	**EA**	08	**C2**	61	**49**	3B	**DC**	2F	**13**	6D	**FD**

Cuadro 16: Caja inversa de sustitución logarítmica (invSL-Box)

Además esta función es única, la unicidad es directa de aplicar el resultado de composición de funciones biyectivas, es decir que no hay otra función que sea homologa a ${}_{inv}S_{BOX}$ en términos de los estados que no sea ${}_{inv}LS_{BOX}$.

Ahora ya obtenidas las tablas logarítmica y antilogarítmica, y aplicando las caracterizaciones 5 y 6 para determinar una función de transición, es posible definir un AC transformador en GF(2^8) en términos del espacio de estados E_{AC} determinado por la primitiva criptográfica (cifrado ó descifrado) y las cajas LS-Box e invLS-Box como sigue: Consideremos una configuración de vecindad de radio cero pues la transformación affin no esta en el contexto de ningún otro estado más que el del estado central, de un lattice unidimensional, con conjunto estados correspondientes a los de la tabla 13, y con función de transición determinada por la siguiente expresión.

$$\top(x_{c_t}) = \begin{cases} x_{c_{t+1}} = LS_{BOX}(x_{c_t}) & si \quad AC \quad Configurado \quad en \quad modo \quad cifrador \\ x_{c_{t+1}} =_{inv} LS_{BOX}(x_{c_t}) & si \quad AC \quad Configurado \quad en \quad modo \quad descifrador \end{cases}$$

Donde x_{c_t} representa el valor de la célula central en el instante t y $x_{c_{t+1}}$ su valor en el siguiente instante.

Teniendo como precedente el modelado matemático anteriormente desarrollado y una vez determinada la función de transición del autómata celular transformador y su inverso en GF 2^8, es momento de implementar la transformación affin En la presente sección se muestra con algunos gráficos, la implementación de la transformación affin del sistema.

Las siguientes gráficas muestran que el autómata celular, presentan ciertos comportamientos que se asemejan a la clasificación de Wolfram, con algunas entradas especificas:

A continuación se describen las entradas del sistema para cada uno de los casos.

Prueba: Clase II

La evolución del autómata celular lleva a un conjunto de estructuras simples que son estables o periódicas.

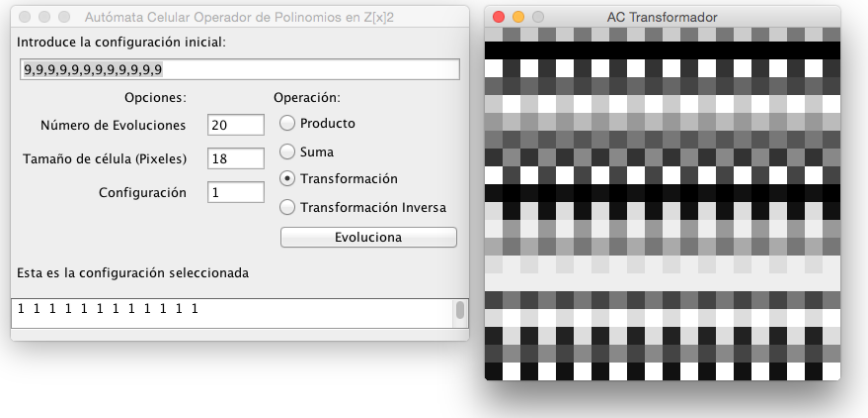

Figura 49: Ejemplo de cálculo clase II. Para el AC transformador

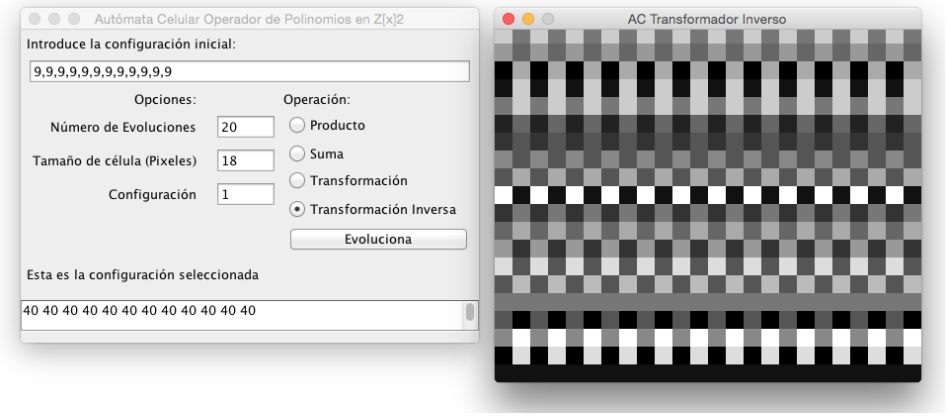

Figura 50: Ejemplo de cálculo clase II. Para el AC transformador inverso

Prueba: Clase III

La evolución del autómata celular lleva a un conjunto siempre caótico.

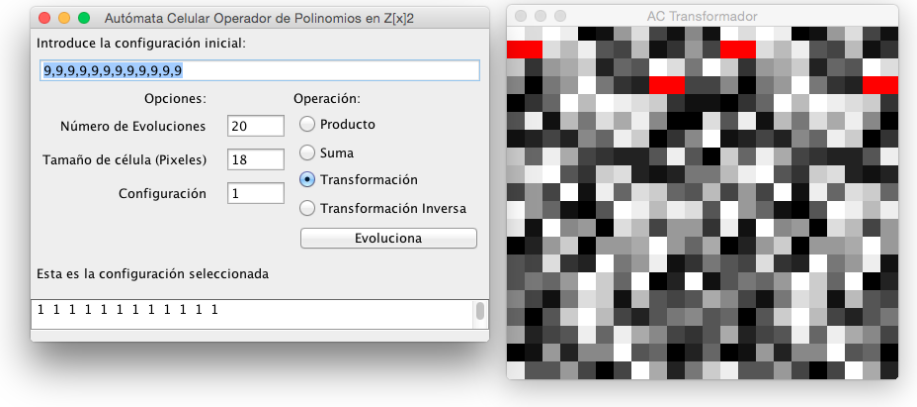

Figura 51: Ejemplo de cálculo clase III. Para el AC transformador

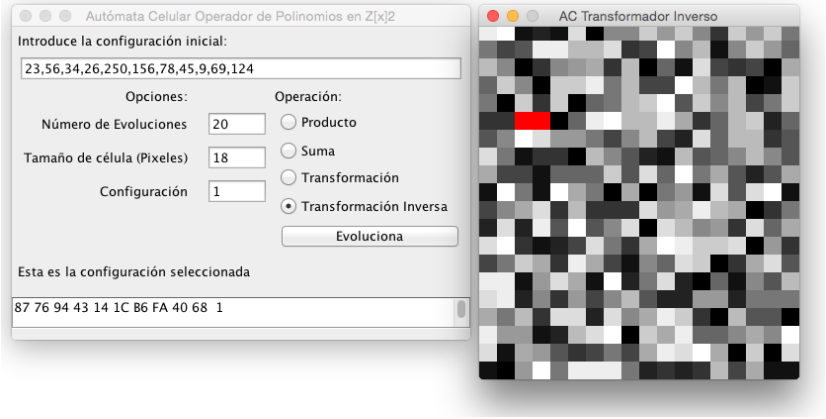

Figura 52: Ejemplo de cálculo clase III. Para el AC transformador inverso

5.1.3. Autómata celular sumador

La tercera operación primitiva a ser caracterizada es la suma en GF(2^8), en general resolver el problema de la suma en términos de logaritmos en cualquier campo incluyendo GF(2^8), es una tarea sumamente difícil, pues no existe una expresión en términos de los logaritmos

para caracterizar la suma, pues la ecuación $Log_b(a+c) = r$ no tiene expresión conocida en términos de los logaritmos, dicho problema se ha abordado desde distintos puntos de vista de la matemática y no se ha podido dar una expresión general, efectiva y simple que permita resolver la ecuación, esto se debe por la forma en que se definen las operaciones de suma y producto en cada campo. En el presente trabajo terminal se logró resolver el problema únicamente para el campo GF(2^8), dicha operación fue la operación más complicada de caracterizar en términos de los estados del autómata, pues la compuerta Xor (equivalente a la suma coeficiente a coeficiente en GF(2^8)), es altamente sensible las condiciones de sus operandos.
Antes de comenzar la caracterización es importante definir la siguiente función h, pues sin ella la caracterización no seria posible, sea $h : GF(2^8) \to GF(2^8)$ tal que si $p(x) \quad GF(2^8), \quad h(p(x)) = p(x) + 1$.

Observación: h cumple las siguientes propiedades:

- h Es involutiva, en otras palabras la inversa h de ella misma h:
  No es difícil demostrarlo, pues si componemos h seguida de h, se conserva la cerradura pues:
  $h \circ h : GF(2^8) \to GF(2^8) \to GF(2^8)$, implica $h \circ h : GF(2^8) \to GF(2^8)$, entonces si $p(x) \in GF(2^8)$ $h(p(x)) = p(x) + 1$ tenemos que:

$$h(h(p(x))) = h(p(x)+1) = (p(x)+1)+1 = p(x)+2 = p(x)$$

  lo que implica que h es involutiva en GF(2^8) y por lo tanto $h^{-1} = h$.

- h Biyectiva, esto se cumple directo de la propiedad anterior.

- h Es una función lineal en GF(2^8), directo de la definición de h.

- h Es la única función lineal en GF(2^8)→GF(2^8) de la forma $p(x)+K$ distinta de la identidad, pues para toda función de la forma $g(p(x)) = p(x) + k$ si $k = 2n$ con n entero, entonces $g(p(x)) = p(x) + 2n = p(x) + 0 = id(p(x))$ si $k = 2n + 1$ para algún n entero, entonces $g(p(x)) = p(x) + 2n + 1 = p(x) + 0 + 1 = p(x) + 1 = h(p(x))$ y por lo tanto $g = h$, y no hay más casos pues las constantes solo pueden ser enteros, por que si no fuera así, no estaríamos en el campo GF(2^8) .

El siguiente paso consiste en caracterizar h en términos de los estados del autómata, al igual que se hizo para las cajas de sustitución.

Caracterización 7: sobre la función lineal h en GF(2^8) y el espacio de estados
Sean $e_1 \quad y \quad e_2$ imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces existe una función biyectiva $H : E_{AC} \to E_{AC}$ tal que $h[p_1(x)] = p_2(x) \in$ GF(2^8) $\Leftrightarrow H(e_1) = e_2 \in E_{AC}$.

Demostración:
La construcción es análoga a las dos caracterizaciones anteriores, estableciendo el diagrama.

$$\begin{array}{ccc} E_{AC} & \xrightarrow{\quad\exists H\quad} & E_{AC} \\ \downarrow A_{BOX} & & \uparrow L_{BOX} \\ GF(2^8) & \xrightarrow{\quad h\quad} & GF(2^8) \end{array}$$

Consideremos las funciones $A_{BOX} : E_{AC} \to GF(2^8)$ y $h : GF(2^8) \to GF(2^8)$, que figuran en el diagrama, ambas son biyectivas pues A_{BOX} es la inversa de L_{BOX} y h es la función lineal definida anteriormente, por lo tanto la composición A_{BOX} seguida de h existe y es biyectiva, de forma que:

$$h \circ A_{BOX} : E_{AC} \to GF(2^8)$$

Consideremos la tercera función que figura en el diagrama $L_{BOX} : GF(2^8) \rightarrow E_{AC}$ que es biyectiva, y realicemos la composición de funciones L_{BOX} seguida de $h \circ A_{BOX}$ que de igual manera existe y es biyectiva.

$$L_{BOX} \circ h \circ A_{BOX} : E_{AC} \rightarrow E_{AC}$$

Aseguramos que dicha función es la H que hace conmutar al diagrama anterior y por lo tanto será la función homologa a h en términos de los estados del Autómata Celular. Para probar esto es necesario partir la demostración en dos implicaciones:

$\Rightarrow$: Dado que existe $H := L_{BOX} \circ h \circ A_{BOX}$ y $h(p_1(x)) = p_2(x) \Rightarrow f(e_1) = e_2$.

$$\begin{aligned} \Rightarrow f(e_1) &= L_{BOX} \circ h \circ A_{BOX}[e_1] \quad por \quad hipotesis \\ &= L_{BOX} \circ h[A_{BOX}[e_1]] \\ &= L_{BOX} \circ h[p_1(x)] \quad por \quad hipotesis \\ &= L_{BOX}[h[p_1(x)]] \\ &= L_{BOX}[p_2(x)] \quad por \quad hipotesis \\ L_{BOX}[h[A_{BOX}[e_1]]] &= e_2 \\ H(e_1) &= e_2 \end{aligned}$$

$\Leftarrow$: Dado que existe $H = L_{BOX} \circ h \circ A_{BOX}$ y $H(e_1) = e_2 \Rightarrow h(p_1(x)) = p_2(x)$.

$$\begin{aligned} \Rightarrow f(e_1) &= e_2 \quad por \quad hipotesis \\ L_{BOX}[h[A_{BOX}[e_1]]] &= e_2 \\ A_{BOX}[L_{BOX}[h[A_{BOX}[e_1]]]] &= A_{BOX}[e_2] \\ h[A_{BOX}[e_1]] &= p_2(x) \\ h[p_1(x)] &= p_2(x) \end{aligned}$$

Por lo tanto la función H existe y es la homologa la función h para GF(2^8). Además esta función es única, la unicidad es directa de aplicar el resultado de composición de funciones biyectivas, es decir que no hay otra función que sea homologa a h en términos de los estados que no sea H. Notemos que el diagrama ofrece una forma sencilla de calcular la función, aplicando directamente la formula $L_{BOX}[h[A_{BOX}[e_1]]]$ para todo e_1 en la tabla de estados, dicho cálculo no requiere mayor problema y se muestra a continuación.

H	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	FF	**19**	32	**DF**	64	**8A**	BF	**70**	C8	**78**	15	**F5**	7F	**63**	E0	**21**
1	91	**44**	F0	**5C**	2A	**0A**	EB	**C4**	FE	**01**	C6	**68**	C1	**B5**	42	**2D**
2	23	**0F**	88	**20**	E1	**B3**	B8	**6A**	54	**9D**	14	**79**	D7	**1F**	89	**65**
3	FD	**C5**	02	**EE**	8D	**93**	D0	**3F**	83	**53**	6B	**52**	84	**BA**	5A	**37**
4	46	**A2**	1E	**D8**	11	**82**	40	**6D**	C3	**EC**	67	**C7**	71	**E4**	D4	**AE**
5	A8	**A0**	3B	**39**	28	**AA**	F2	**A7**	AF	**CB**	3E	**D1**	13	**9E**	CA	**B0**
6	FB	**BE**	8B	**0D**	04	**2F**	DD	**4A**	1B	**F8**	27	**3A**	A1	**47**	7E	**F6**
7	07	**4C**	A6	**F3**	D6	**7A**	A4	**99**	09	**2B**	75	**B7**	B4	**C2**	6E	**0C**
8	8C	**EF**	45	**38**	3C	**FA**	B1	**90**	22	**2E**	05	**62**	80	**34**	DA	**96**
9	87	**10**	D9	**35**	CE	**BC**	8F	**B2**	E2	**77**	C9	**9F**	A9	**29**	5D	**9B**
A	51	**6C**	41	**B6**	76	**E3**	72	**57**	50	**9C**	55	**D3**	E5	**E8**	4F	**58**
B	5F	**86**	97	**25**	7C	**1D**	A3	**7B**	26	**F9**	3D	**CC**	95	**DB**	61	**06**
C	F7	**1C**	7D	**48**	17	**31**	1A	**4B**	08	**9A**	5E	**59**	BB	**CF**	94	**CD**
D	36	**5B**	F1	**AB**	4E	**E9**	74	**2C**	43	**92**	8E	**BD**	FC	**66**	ED	**03**
E	0E	**24**	98	**A5**	4D	**AC**	E7	**E6**	AD	**D5**	F4	**16**	49	**DE**	33	**81**
F	12	**D2**	56	**73**	EA	**0B**	6F	**C0**	69	**B9**	85	**60**	DC	**30**	18	**00**

Cuadro 17: Función H

La función H cumple las siguientes propiedades importantes:

1. H es involutiva, es decir que H es su propia inversa, i. e. $H^{-1} = H$.

$$\begin{aligned} H &= L_{BOX} \circ h \circ A_{BOX} \\ (H)^{-1} &= (L_{BOX} \circ h \circ A_{BOX})^{-1} \\ &= (A_{BOX}^{-1}) \circ (L_{BOX} \circ h)^{-1} \\ &= L_{BOX} \circ (L_{BOX} \circ h)^{-1} \\ &= L_{BOX} \circ (h^{-1} \circ L_{BOX}^{-1}) \\ H^{-1} &= L_{BOX} \circ h \circ A_{BOX} = H \end{aligned}$$

2. Si $h[p_1(x)] = p_2(x)$ y $H[e_1] = e_2$ entonces $H = L_{BOX} \circ h$.

$$\begin{aligned} h[p_1(x)] &= p_2(x) \\ L_{BOX}[h[p_1(x)]] &= L_{BOX}[p_2(x)] \\ L_{BOX}[h[p_1(x)]] &= e_2 \\ L_{BOX} \circ h[p_1(x)] &= H[e_1] \Rightarrow L_{BOX} \circ h \quad = \quad H \end{aligned}$$

Ya se cuentan con todos los elementos para poder caracterizar la suma en términos de los estados del Autómata celular.

Caracterización 8: sobre la suma en GF(2^8) y el espacio de estados
Sean $FF \neq e_1 \neq e_2 \neq FF$ imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces $p_1(x) + p_2(x) \in$ GF(2^8) $\Leftrightarrow Log_{g(x)}[p_1(x) + p_2(x)] \in E_{AC} - \{FF\}$.

Demostración:
Sabemos que si e_1 y e_2 son elementos de E_{AC}, entonces son estados de la tabla 13, por hipótesis como ambos estados $e_1 \neq FF$ y $e_2 \neq FF$, implica que existen $e_1 = n$ y $e_2 = m$ para algún n y m $\in \quad Z_{255}$ tales que $g(x)^n$, $g(x)^m \in$ GF(2^8)$_{/m(x)}$ donde $g(x) = (x+1)$.

$$
\begin{aligned}
p_1(x)+p_2(x) &= g(x)^n + g(x)^m \\
&= g(x)^n \bullet (1 + g(x)^{m-n}) \\
&= g(x)^n \bullet (g(x)^m \bullet g(x)^{-n} + 1) \\
p_1(x)+p_2(x) &= g(x)^n \bullet \left(g(x)^m \bullet \left(\frac{1}{g(x)^n} \right) + 1 \right) \\
Log_{g(x)}[p_1(x)+p_2(x)] &= Log_{g(x)} \left[g(x)^n \bullet \left(g(x)^m \bullet \left(\frac{1}{g(x)^n} \right) + 1 \right) \right] \\
Log_{g(x)}[p_1(x)+p_2(x)] &= L_{BOX} \left[g(x)^n \bullet \left(g(x)^m \bullet \left(\frac{1}{g(x)^n} \right) + 1 \right) \right] \\
&= L_{BOX}\left[g(x)^n\right] + L_{BOX} \left[\left(g(x)^m \bullet \left(\frac{1}{g(x)^n} \right) \right) + 1 \right]_{por \quad C1} \\
&= L_{BOX}\left[g(x)^n\right] + L_{BOX} \left[A_{BOX} \left[L_{BOX} \left(g(x)^m \bullet \left(\frac{1}{g(x)^n} \right) \right) \right] + 1 \right] \\
&= L_{BOX}\left[g(x)^n\right] + L_{BOX} \left[A_{BOX} \left[L_{BOX}\left(g(x)^m\right) + L_{BOX} \left(\frac{1}{g(x)^n} \right) \right] + 1 \right]_{por \quad C1} \\
&= L_{BOX}\left[g(x)^n\right] + L_{BOX} \left[A_{BOX} \left[L_{BOX}\left(g(x)^m\right) + (-n) \mod 255 \right] + 1 \right]_{por \quad C4} \\
&= L_{BOX}\left[g(x)^n\right] + L_{BOX} \left[A_{BOX} \left[(m-n)_{mod \quad 255} \right] + 1 \right]_{por \quad C4} \\
&= n + L_{BOX}\left[h\left(A_{BOX} \left[(m-n)_{mod \quad 255} \right] \right) \right] \\
&= n + H[(m-n)_{mod \quad 255}]_{por \quad C7} \\
Log_{g(x)}[p_1(x)+p_2(x)] &= e_1 + H[e_1 - e_2] \quad \in \quad E_{AC}
\end{aligned}
$$

El resultado obtenido es bastante notable puesto que permite calcular la suma de estados en términos de los estados del autómata, mediante la formula $e_1 + H[e_1 - e_2]$ que es solución al problema de la suma de los logaritmos para GF(2^8), que como ya se había mencionado anteriormente es difícil de resolver.

Caracterización 9: Sobre los inversos aditivos de GF(2^8) y el espacio de estados
Sean las imágenes de e_1 y e_2 elementos del espacio de estados E_{AC}, bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces $p_1(x) + p_2(x) = 0 \in$ GF(2^8) si y solo si $e_1 = e_2$.

Demostración:
Como de e_1 y e_2 no sabemos nada, si no únicamente que por hipótesis $p_1(x) + p_2(x) = 0$, entonces por un lado $p_1(x) = p_2(x) \Rightarrow L_{BOX}[p_1(x)] = L_{BOX}[p_2(x)]$ y por otro como:

$$
\begin{aligned}
p_1(x) + p_2(x) &= 0 \\
L_{BOX}[p_1(x) + p_1(x)] &= L_{BOX}[0] \quad \in \quad E_{AC} - Z_{255} \\
L_{BOX}[p_1(x) + p_1(x)] &= FF \quad pero \\
L_{BOX}[p_1(x)] &= L_{BOX}[p_2(x)] \\
e_1 &= e_2
\end{aligned}
$$

Con este resultado queda definida la suma en términos de estados de GF(2^8) cuando los estados son iguales, es decir que si dos estados son iguales entonces la suma de sus preimagenes fue 0 y por lo tanto en la siguiente instante el AC debe pasar a FF.

Caracterización 10: sobre el neutro aditivo de GF(2^8) y el espacio de estados
Sean las imágenes de $e_1 \neq e_2$ elementos del espacio de estados E_{AC}, bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, entonces $p_1(x) = 0$ ó $p_2(x) = 0$ si y solo si $A_{BOX}[Mine_1, e_2] = p_1(x) + p_2(x)$.

Demostración:
Como $e_1 \neq e_2$ entonces $p_1(x) + p_2(x) \neq 0 \Rightarrow$ y por lo tanto:

$$\begin{aligned}
p_1(x) + p_2(x) = p_1(x) \quad & o \quad p_1(x) + p_2(x) = p_2(x) \\
A_{BOX}[p_1(x) + p_2(x)] = A_{BOX}[p_1(x)] \quad & o \quad A_{BOX}[p_1(x) + p_2(x)] = A_{BOX}[p_2(x)] \\
A_{BOX}[p_1(x) + p_2(x)] = e_1 \quad & o \quad A_{BOX}[p_1(x) + p_2(x)] = e_2 \\
A_{BOX}[p_1(x) + p_2(x)] = Min\{e_1, FF\} \quad & o \quad A_{BOX}[p_1(x) + p_2(x)] = Min\{e_2, FF\} \\
A_{BOX}[p_1(x) + p_2(x)] \quad & = \quad Min\{e_1, e_2\}
\end{aligned}$$

Con este último resultado quedan abarcados todos los casos de la suma en GF(2^8) pues indica que si se reciben dos estados con uno de ellos FF indica que se realizo una suma en GF(2^8) con el cero y por lo tanto debemos quedarnos con el mínimo de ambos para preservar e neutro aditivo del campo.

Caracterización 11: Sobre la conmutatividad de la suma GF(2^8) y el espacio de estados
Si e_1 y e_2 son imágenes bajo L-box de $p_1(x)$ y $p_2(x)$ correspondientemente, si $p_1(x) + p_2(x) = p_2(x) + p_1(x) \in$ GF(2^8) si y solo si $L_{Box}[p_1(x) + p_2(x)]$=$L_{Box}[p_2(x) + p_1(x)]$.

Demostración:
Tenemos 3 casos:

Caso 1 Cuando $FF \neq e_1 \neq e_2 \neq FF$, la formula $e_1 + H[e_1 - e_2]$ tiene sentido, ya que $e_1 + H[e_1 - e_2] = Log_{g(x)}(p_1(x) + p_2(x)) = Log_{g(x)}(p_2(x) + p_1(x)) = e_2 + H[e_2 - e_1]$. En conclusión $e_1 + H[e_1 - e_2] = e_2 + H[e_2 - e_1]$, preservando la conmutatividad del campo GF(2^8) en términos del estado del autómata.

Caso 2 Cuando $e_1 = e_2$ entonces $p_1(x) + p_2(x) = 0 = p_1(x) + p_2(x)$ y por lo tanto $L_{BOX}(p_1(x) + p_2(x)) = L_{BOX}(p_1(x) + p_2(x)) = FF$

Caso 3 Cuando $A_{BOX}[Min\{e_1, e_2\}] = p_1(x) + p_2(x) = p_2(x) + p_1(x) = A_{BOX}[Min\{e_2, e_1\}]$.

Autómata Celular Sumador en GF(2^8)
Ahora ya obtenidas la función H y caracterizaciones 11, 10, 9, 8 y 7 para determinar una función de transición, es posible definir un AC multiplicador en GF(2^8) en términos del espacio de estados E_{AC} determinado por la función H como sigue: Consideremos una configuración de vecindad de radio un medio por simplicidad, con un lattice unidimensional, con conjunto estados correspondientes a los de la tabla 13, y con función de transición dada por la siguiente expresión.

$$\boxplus(x_{c_t}, x_{d_t}) = \begin{cases} x_{c_{t+1}} = FF & si \quad x_{c_t} = x_{d_t} \\ x_{c_{t+1}} = min\{x_{c_t}, x_{d_t}\} & si \quad x_{c_t} = FF \ o \ x_{d_t} = FF \\ x_{c_{t+1}} = x_{c_t} + H(x_{c_t} - x_{d_t}) & si \quad x_{c_t} \neq x_{d_t} \neq FF \end{cases}$$

Donde x_{c_t} representa el valor la célula central en el instante t, x_{d_t} el valor de la célula adyacente a la derecha en el instante t y $x_{c_{t+1}}$ el valor de la célula central en el instante t+1.

Teniendo como precedente el modelado matemático anteriormente desarrollado y una vez determinada la función de transición del autómata celular sumador en GF 2^8, es momento de implementar la suma polinomial. En la presente sección se muestra con algunos gráficos, la implementación de la suma polinomial del sistema.

Las siguientes gráficas muestran que el autómata celular, presentan ciertos comportamientos que se asemejan a la clasificación de Wolfram, con algunas entradas especificas:

A continuación se describen las entradas del sistema para cada uno de los casos.

Prueba: Clase I

Evolución a un estado uniforme. Después de transcurrido un cierto número de generaciones, todas las células del autómata convergen a un solo estado.

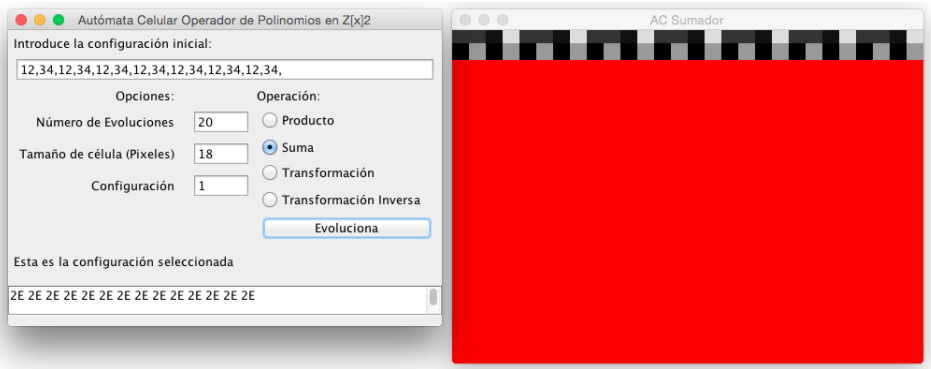

Figura 53: Ejemplo de cálculo clase I para el autómata sumador.

El estado FF se propaga por la lattice si todos todos son inversos aditivos. **Prueba: Clase II para el autómata sumador**

La evolución del autómata celular lleva a un conjunto de estructuras simples que son estables o periódicas.

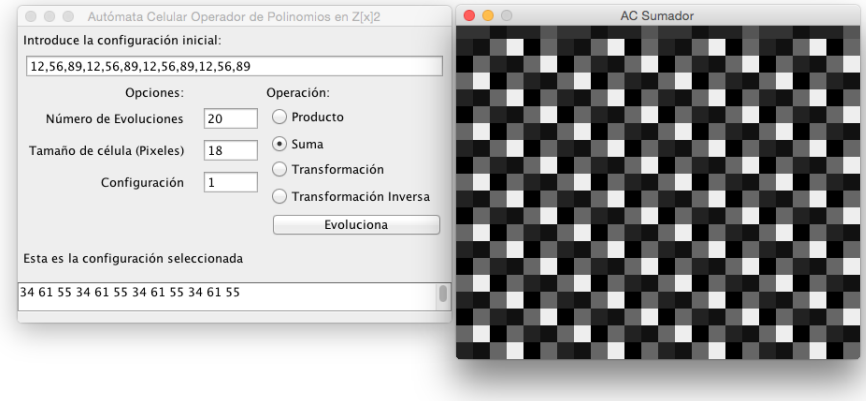

Figura 54: Ejemplo de cálculo clase II para el autómata sumador.

Un patron periodico puede indicar que los elementos de la primera configuración forman una sub estructura algebraica, conocida como subgrupo aditivo.

Prueba: Clase III para el autómata sumador

La evolución del autómata celular lleva a un conjunto siempre caótico.

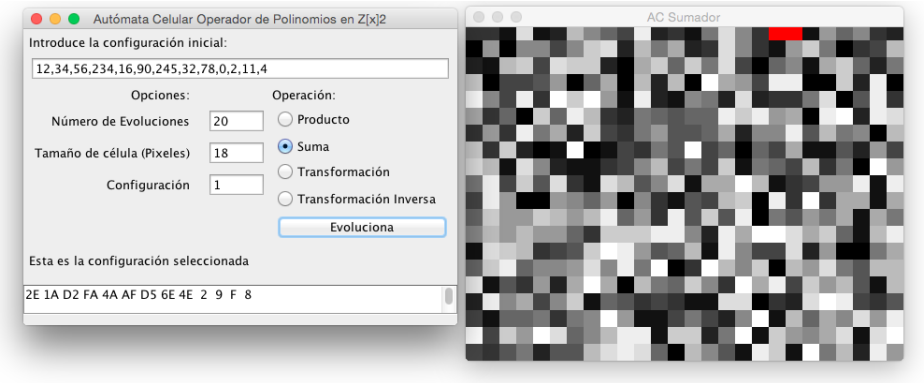

Figura 55: Ejemplo de cálculo clase III para el autómata sumador.

5.2. Caracterización de transformaciones rígidas

Como ya se había mencionado en el algoritmo AES existen tres operaciones primitivas, las denominamos primitivas puesto que son las operaciones más básicas que podemos encontrar en el algoritmo, dichas operaciones actúan directamente sobre las unidades de cifrado de AES, es decir los polinomios (representados por bytes) que pertenecen al campo de Galois GF(2^8). Sin embargo también existen operaciones (funciones) esenciales para el cifrado que no actúan a bajo nivel, es decir no transforman las unidades de cifrado si no más bien disponen de los datos en un orden correcto para posteriormente ser transformados, por lo tanto hasta cierto punto las consideremos transformaciones rígidas (haciendo analogía con las transformaciones rígidas en matemáticas) pues son aplicadas a lo largo del cifrado prescindiendo del valor de los elementos que están transformando, dicho en otras palabras, preservan el valor de las unidades de procesamiento. La razón de definir estas operaciones es que están pensadas para poder considerar un autómata celular de orden 1, sobre las cuales actúen imitando las transformaciones rígidas del algoritmo AES, dichas operaciones son las siguientes.

1. Mezcla (Merge): la mezcla o merge es una transformación que esta diseñada para imitar la disposición de los bytes como columnas vector en una matriz del algoritmo AES y se define como sigue:

DEFINICIÓN 28 (MEZCLADO)
Dados dos conjuntos A y B tales que $|A| = |B|$, ordenados por un conjunto de indices I, se define a la mezcla de los conjuntos como: la yuxtaposición de a_i seguido de b_i en un conjunto C, tales que la posición en a_i en C es ix2, donde $a_i \in A$, $b_i \in B$ y $i \in I$.

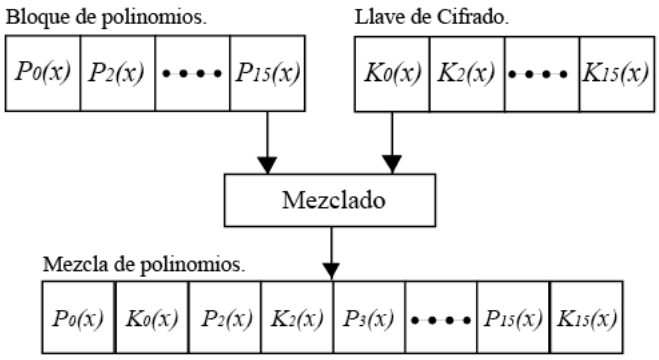

Figura 56: Ejemplo de la mezcla aplicada al bloque de texto plano y llave de entrada.

Observación: La mezcla no esta definida únicamente para estos dos conjuntos puesto que a lo largo del cifrado con el AC se realizaran mezclas con conjuntos cuyos elementos son de distinta naturaleza a los del siguiente ejemplo.

2. Permutación. La permutación es una transformación diseñada para imitar la función de ShiftRows en el algoritmo AES estándar en su versión de 128 bits, esta transformación esta inducida por el orden de los elementos en lattice que define la mezcla. Para construirla consideremos la siguiente matriz y su transformación bajo ShiftRows.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \xrightarrow{ShiftRows} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

Pero por la mezcla de la matriz con algún otro conjunto, si A era el primer argumento si indice i bajo la mezcla esta dado por:

$$\begin{bmatrix} (a_{00})_{0x2} & (a_{01})_{4x2} & (a_{02})_{8x2} & (a_{03})_{12x2} \\ (a_{10})_{1x2} & (a_{11})_{5x2} & (a_{12})_{9x2} & (a_{13})_{13x2} \\ (a_{20})_{2x2} & (a_{21})_{6x2} & (a_{22})_{10x2} & (a_{23})_{14x2} \\ (a_{30})_{3x2} & (a_{31})_{7x2} & (a_{32})_{11x2} & (a_{33})_{15x2} \end{bmatrix} \xrightarrow{ShiftRows} \begin{bmatrix} (a_{00})_{0x2} & (a_{01})_{2x2} & (a_{02})_{8x2} & (a_{03})_{12x2} \\ (a_{11})_{5x2} & (a_{12})_{9x2} & (a_{13})_{13x2} & (a_{10})_{1x2} \\ (a_{22})_{10x2} & (a_{23})_{14x2} & (a_{20})_{2x2} & (a_{21})_{6x2} \\ (a_{33})_{15x2} & (a_{30})_{3x2} & (a_{31})_{7x2} & (a_{32})_{11x2} \end{bmatrix}$$

Lo cual induce una permutación σ y σ^{-1}.

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 5 & 10 & 15 & 4 & 9 & 14 & 3 & 8 & 13 & 2 & 7 & 12 & 1 & 6 & 11 \end{pmatrix}$$

la cual tiene inversa y calculándola se tiene:

$$\sigma^{-1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 13 & 10 & 7 & 4 & 1 & 14 & 11 & 8 & 5 & 2 & 15 & 12 & 9 & 6 & 3 \end{pmatrix}$$

Las permutaciones calculadas imitan las funciones ShiftRows e InvShiftRows.

3. Rotación; es la función rotWord definida para expansor de llaves, se define ya que en el cifrado aparecerá un término constante que rota con cada transición del autómata.

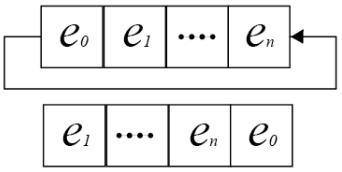

Figura 57: Rotword.

5.3. Construcción del autómata cifrador

El autómata cifrador se construye de la siguiente manera:

- Consideremos un conjunto de estados E_{AC} dados por $L_{BOX} = \mathbb{Z}_{255} \cup \{0xFF\}$.
- Un lattice de orden 1, i.e. de dimensión 1 con lattice periódica.
- Con funciones de transición dadas por la suma ($\boxplus$) , producto ($\boxdot$), la transformación affin ($\top$).
- Vecindades determinadas por cada una de las funciones de transición.

El proceso de cifrado comienza de la siguiente forma: se reciben el bloque de polinomio correspondiente al texto plano y el bloque de polinomios correspondiente a la primera o última llave correspondiente a la primitiva criptografía cifrado ó descifrado, estos bloques se mezclan y se codifican mediante L_{BOX} para obtener un bloque de estados.

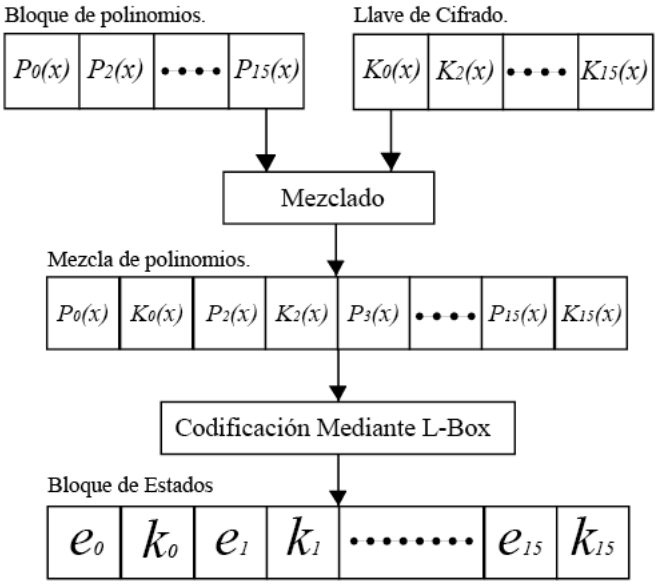

Figura 58: Codificación de un bloque de polinomios a un bloque de estados.

5.3.1. Función AddRoundKey para la lattice

Esta función podemos distinguir dos casos, el primero se trata de la primera adición ya sea para cifrar o descifrar entonces se procede a aplicar la regla de la suma directamente como se muestra a continuación.

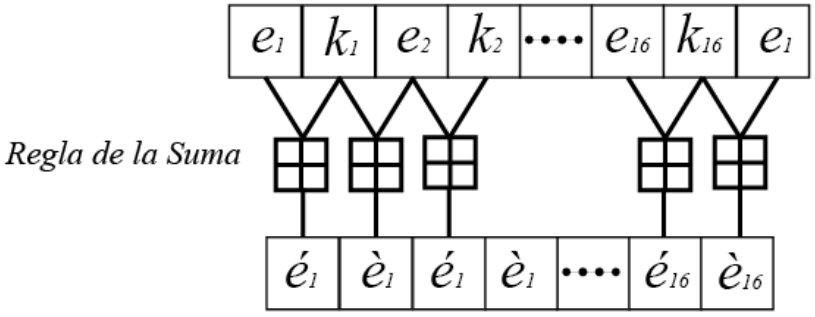

Figura 59: Aplicación de la regla de la suma.

Si se trata de alguna de las 9 adiciones posteriores entonces se procede a expandir la i-ésima llave de la ronda que provee el expansor en términos de los estados del autómata, para posteriormente mezclarla con el bloque de estados de la última transición generada por el autómata para después aplicar la regla de la suma, como se muestra a continuación.

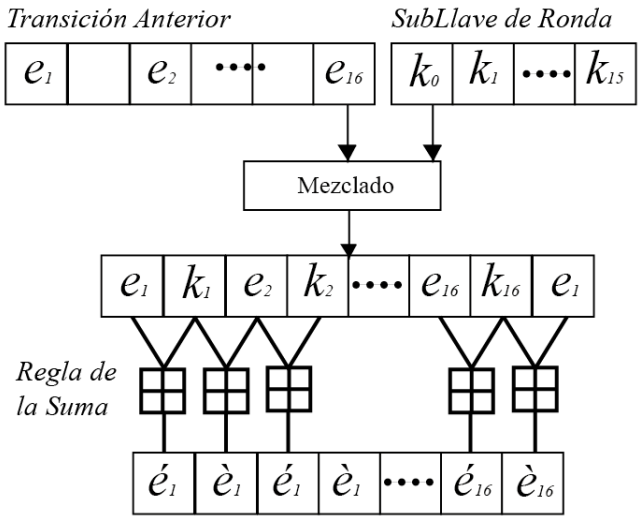

Figura 60: Mezclado y aplicación de la regla de la suma.

5.3.2. Función MixColumnes para lattice

Esta función podemos distinguir los casos de descifrado y cifrado, si el autómata se encuentra en modo cifrador entonces se procede aplicar la permutación (σ), y si se cuenta en modo descifrado se aplica la permutación inversa (σ^{-a}).

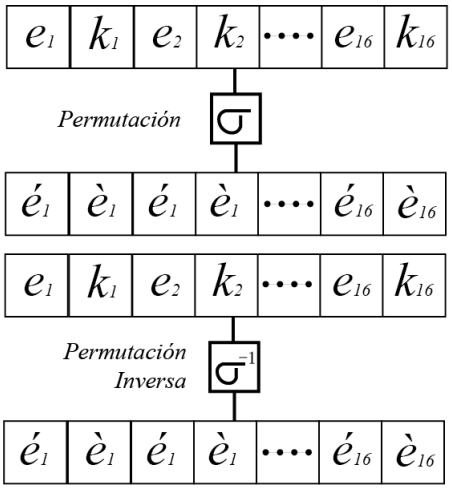

Figura 61: Permutación inducida sobre la lattice.

5.3.3. Función SubBytes para la lattice

Esta función podemos se distinguirle los casos de descifrado y cifrado, si el autómata se encuentra en modo cifrador, entonces se procede a aplicar la regla de la transformación affin ($\top$), y si se encuentra en modo descifrado se aplica la regla de la transformación affin inversa ($\perp$).

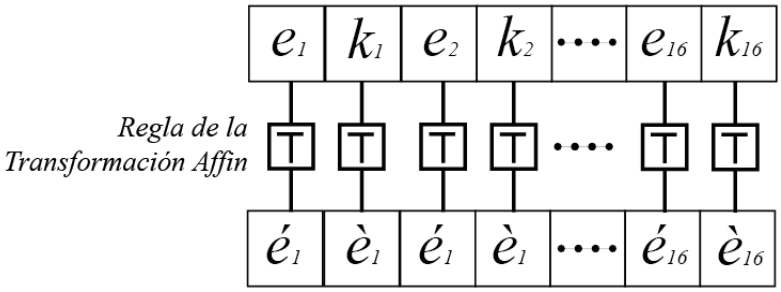

Figura 62: Aplicación de la transformación affin.

5.3.4. Función MixColumns para la lattice

En el producto de matrices del estándar AES figuran dos constates cíclicas C_E y C_D dichas constantes que no son fáciles de observar en el producto matricial, sin embargo su aparición permite expresar el producto de matrices como un producto de elementos cíclicos en una lattice unidimensional, esto se debe a que las constantes de cifrado en el estándar AES son matrices circulantes lo cual induce patrones cíclicos a lo largo del cifrado, para poder imitar la función MixColumns se realiza lo siguiente.

- Se elije la constante dependiendo del modo en el que se encuentra configurado el autómata celular, si se encuentra en modo cifrado se selecciona C_E y si se encuentra configurado en modo descifrado se selecciona C_D. donde:

$$C_E = (19, 00, 00, 01)$$

$$C_D = (DF, 68, EE, C7)$$

- Se crea un vector de 16 elementos concatenando las constantes.
- Se Mezcla con la transición anterior y se aplica la regla del producto $\boxdot$, posteriormente se aplican dos evoluciones con la regla de la suma $\boxplus$.
- Se toma el valor central y se asigna en la posición $ix4$, donde $i \in \{0, 1, 2, 3\}$.
- Se aplica una rotación a la constante y se repite el proceso 3 veces.

Dicho proceso se muestra a continuación.

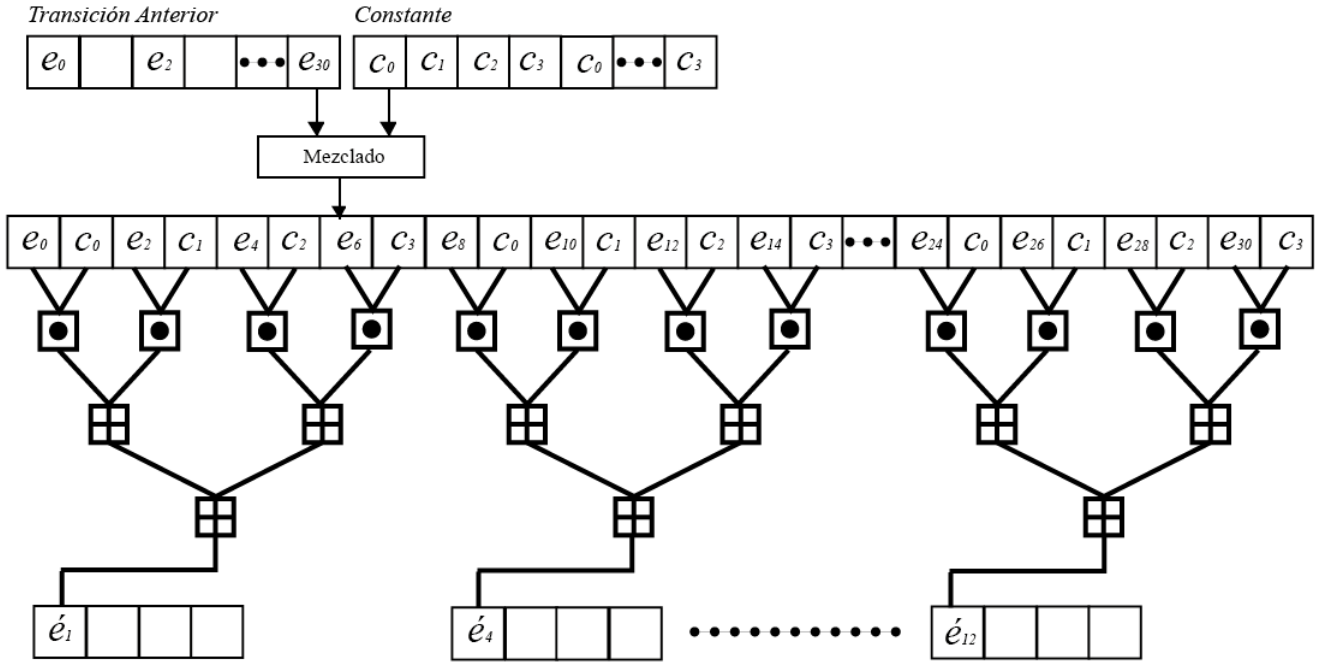

Figura 63: Rotación cero.

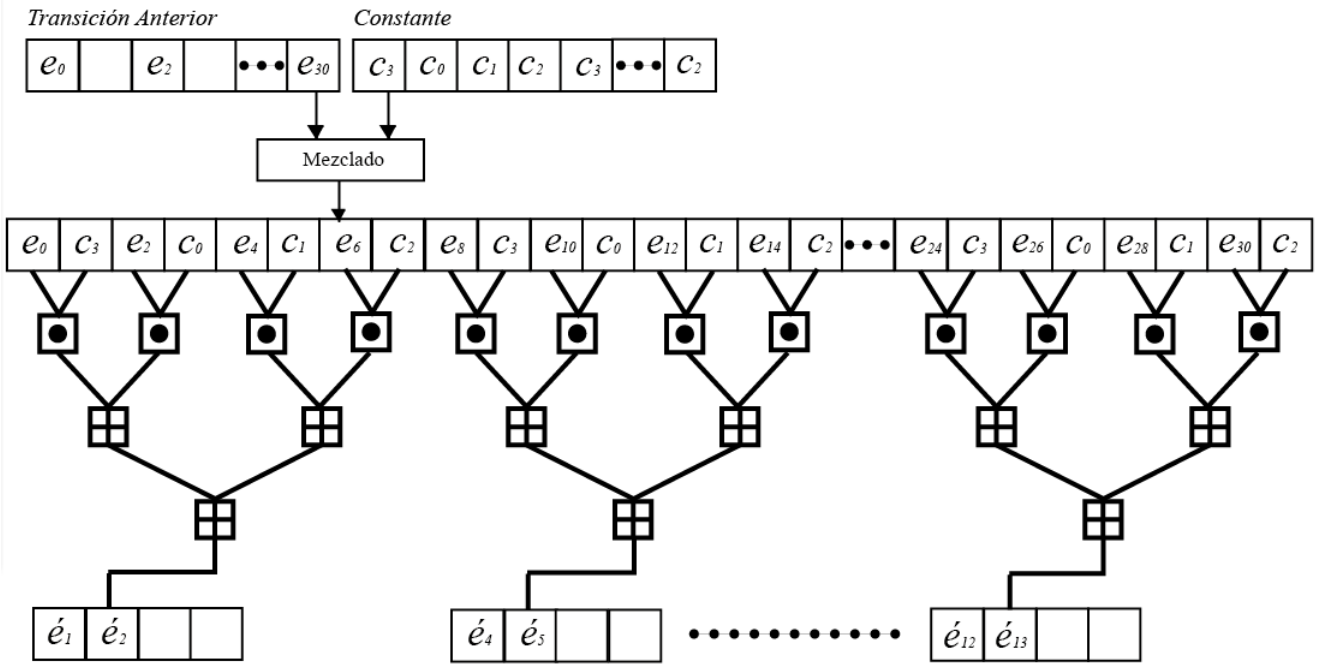

Figura 64: Primera rotación.

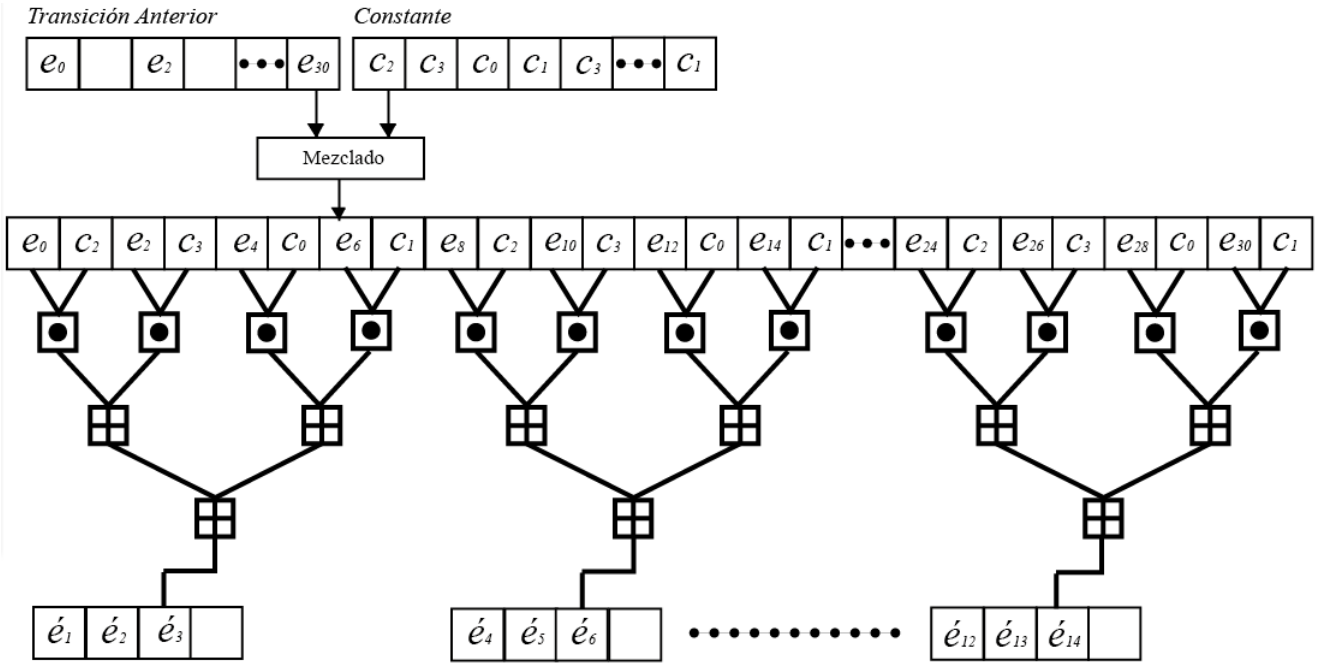

Figura 65: Segunda rotación.

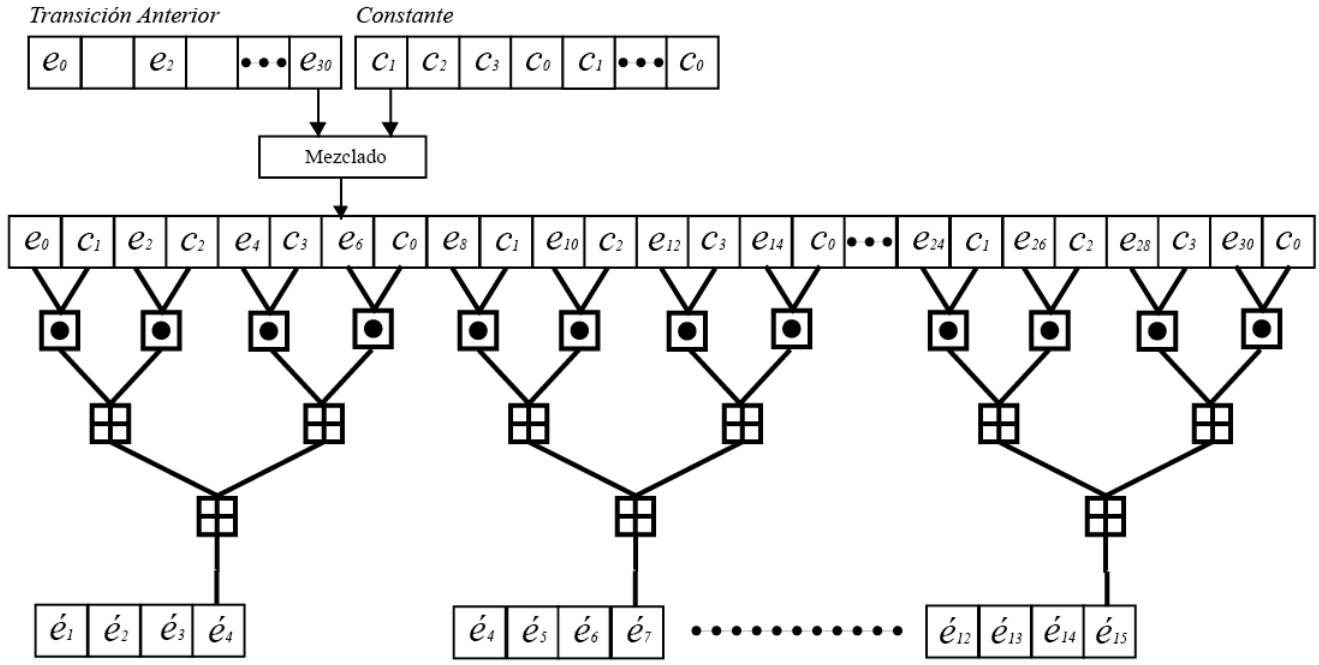

Figura 66: Tercera rotación.

Sin embargo no es necesario hacer cuatro desplazamientos cíclicos sobre la lattice pues por ser periódica ya contiene un desplazamiento cíclico implícito en su estructura, de este modo solo con una rotación se alcanzan todos los productos necesarios para MixColumns, las figuras 67 y 68 muestran este proceso.

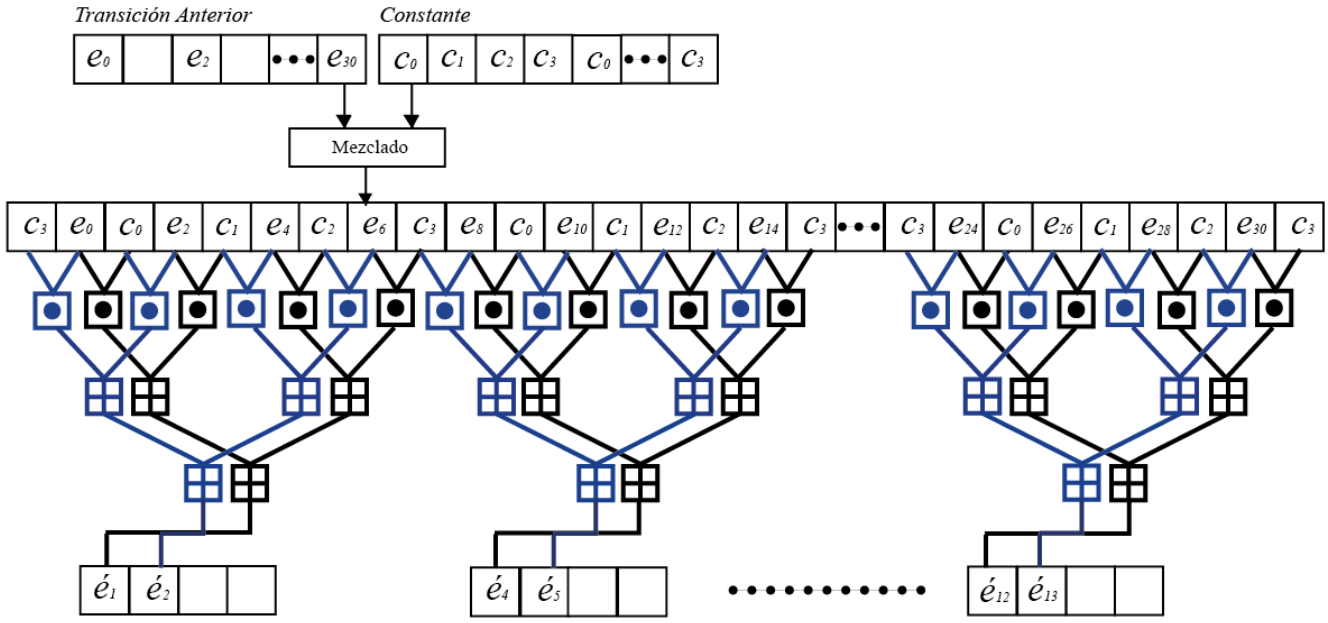

Figura 67: Cálculo en paralelo de la primera y segunda rotación

Sin ningun desplazamiento cíclico se calculan los valores de la primera y segunda rotación.

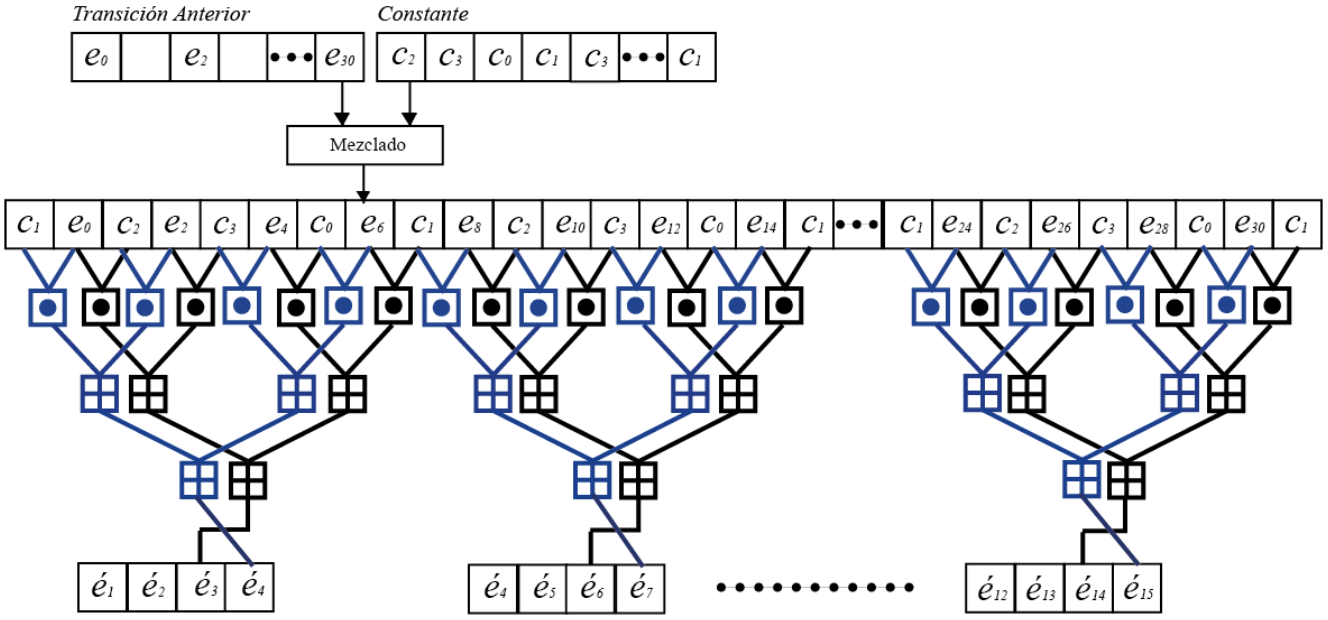

Figura 68: Cálculo en paralelo de la tercera y cuarta rotación.

Con un desplazamiento cíclico se calculan los valores de la tercera y cuarta rotación.
El proceso de cifrado/descifrado para un bloque es similar al algoritmo AES pues se han caracterizado todas las operaciones y construidos todas las funciones en términos de estados y funciones de transición de autómata celular. Para el proceso de cifrado/descifrado únicamente es necesario aplicar 10 rondas de las construcciones anteriormente definidas, con excepción de no aplicar la construcción para mixcolumns en la ultima ronda, de esta manera se obtendrá una cifra equivalente a AES-128 bits codificada en la lattice del autómata celular, para obtener la cifra equivalente solo será necesario decodificar la lattice mediante A_{BOX} y observaremos que las células pares son las que contienen las unidades de procesamiento cifradas.

5.4. Pruebas de cifrado/descifrado

Para dar muestras de que el autómata provee cifras equivalentes al algoritmo AES-128 bits, es necesario establecer dos pruebas, de manera que sea indistinto cifrar / descifrar con cualquiera de las dos entidades, puesto que la caracterización se hizo desde las operaciones que actúan a más bajo nivel hasta aquellas que disponen los elementos para su procesamiento.

5.4.1. Cifrado de un bloque

La primera prueba consiste en cifrar un bloque mediante el autómata y descifrarlo con AES-128 bits, de manera que la entrada y la salida deben ser exactamente las mismas, este proceso lo ilustra la imagen 69.

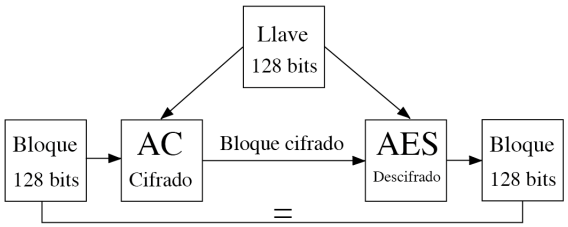

Figura 69: Cifrado de un bloque con AC y descifrado con AES.

Los pasos siguientes se muestran los valores de la lattice y la matriz a lo largo del cifrado para longitud de bloque y clave de cifrado de 16 bytes. Este ejemplo es tomado de estándar AES, FIPS PUB-197 página 33 [18].

Entrada = 3 2 4 3 F 6 A 8 8 8 5 A 3 0 8 D 3 1 3 1 9 8 A 2 E 0 3 7 0 7 3 4
Llave de cifrado = 2 B 7 E 1 5 1 6 2 8 A E D 2 A 6 A B F 7 1 5 8 8 0 9 C F 4 F 3 C

Mezclando:
3 2 2 B 4 3 7 E F 6 1 5 A 8 1 6 8 8 2 8 5 A A E 3 0 D 2 8 D A 6 3 1 A B 3 1 F 7 9 8 1 5 A 2 8 8 E 0 9 3 7 C F 7 4 F 3 4 3 C

Codificando a bloque de estados mediante L_{Box} obtenemos la configuración inicial del autómata celular:
8 A 7 2 B D A 7 F E 8 D D 8 8 1 4 F 4 D E 2 7 B 6 5 8 4 E 6 4 9 2 F 2 D 2 F 1 8 5 9 8 D F 6 4 F 4 4 C 7 2 4 D 1 C 6 3 8 2 1 3 5

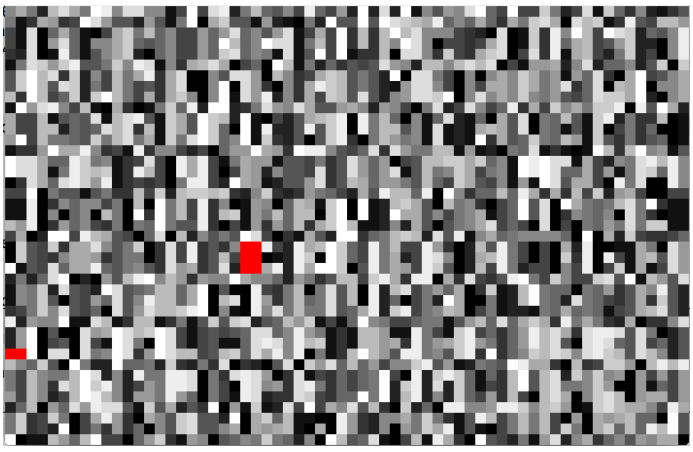

Figura 70: Proceso de cifrado de un bloque de 128 bits con el autómata celular

Al cabo de 41 evoluciones el autómata celular obtiene la cifra equivalente a AES aplicando las reglas caracterizadas, como se muestra en la siguiente figura 70, la configuración final obtenida en la lattice es:

F 0 1 1 B 9 6 B F 4 4 0 6 9 C 3 1 9 8 0 5 6 8 7 C 7 6 2 8 0 F B 5 6 D 1 0 4 B 1 E A 3 0 F 5 4 2 7 1 9 D 2 8 A 3 6 8 5 9 8 A A 0

Decodificando la configuración final del autómata celular mediante A_{Box} obtenemos una configuración que tiene incrustada la cifra equivalente a AES en sus células pares:

3 9 E 1 2 5 6 9 8 4 4 C 1 D 6 3 2 F B D C C 2 9 5 7 F B B 4 D C C F 1 1 B 6 8 5 5 3 9 7 6 7 1 9 B F 6 A 6 4 B 9 8 3 2 9 F

Conservando las células pares obtenemos:

3 9 2 5 8 4 1 D 0 2 D C 0 9 F B D C 1 1 8 5 9 7 1 9 6 A 0 B 3 2

Se procede a descifrar con el algoritmo AES en su versión de 128 bits con la décima expansión de la clave de cifrado:

Ronda	InvShiftRows	InvSubBytes	SubClave	AddRoundKey	InvMixColumns
Entrada	39 02 DC 19 25 DC 11 $6A$ 84 09 85 $0B$ $1D$ FB 97 32		$D0$ $C9$ $E1$ $B6$ 14 EE $3F$ 63 $F9$ 25 $0C$ $0C$ $A8$ 89 $C8$ $A6$	$E9$ CB $3D$ AF 31 32 $2E$ 09 $7D$ $2C$ 89 07 $B5$ 72 $5F$ 94	
Ronda 1	$E9$ CB $3D$ AF 09 31 32 $2E$ 89 07 $7D$ $2C$ 72 $5F$ 94 $B5$	EB 59 $8B$ $1B$ 40 $2E$ $A1$ $C3$ $F2$ 38 13 42 $1E$ 84 $E7$ $D2$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	47 40 $A3$ $4C$ 37 $D4$ 70 $9F$ 94 $E4$ $3A$ 42 ED $A5$ $A6$ BC	87 $F2$ $4D$ 97 $6E$ $4C$ 90 EC 46 $E7$ $4A$ $C3$ $A6$ $8C$ $D8$ 95
Ronda 2	87 $F2$ $4D$ 97 EC $6E$ $4C$ 90 $4A$ $C3$ 46 $E7$ $8C$ $D8$ 95 $A6$	EA 04 65 85 83 45 $5D$ 96 $5C$ 33 98 $B0$ $F0$ $2D$ AD $C5$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	00 $B1$ 54 FA 51 $C8$ 76 $1B$ $2F$ 89 $6D$ 99 $D1$ FF CD EA	BE $D4$ $0A$ DA $3B$ $E1$ 64 83 $D4$ $F2$ $2C$ 86 FE $C8$ $C0$ $4D$
Ronda 3	BE $D4$ $0A$ DA 83 $3B$ $E1$ 64 $2C$ 86 $D4$ $F2$ $C8$ $C0$ $4D$ FE	$5A$ 19 $A3$ $7A$ 41 49 $E0$ $8C$ 42 DC 19 04 $B1$ $1F$ 65 $0C$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	14 46 27 34 15 16 46 $2A$ $B5$ 15 56 $D8$ BF EC $D7$ 43	$F7$ 27 $9B$ 54 83 43 $B5$ AB 40 $3D$ 31 $A9$ $3F$ $F0$ FF $D3$
Ronda 4	$F7$ 27 $9B$ 54 AB 83 43 $B5$ 31 $A9$ 40 $3D$ $F0$ FF $D3$ $3F$	26 $3D$ $E8$ FD $0E$ 41 64 $D2$ $2E$ $B7$ 72 $8B$ 17 $7D$ $A9$ 25	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	$4B$ $2C$ 33 37 86 $4A$ $9D$ $D2$ $8D$ 89 $F4$ 18 $6D$ 80 $E8$ $D8$	$A1$ 78 10 $4C$ $4F$ $E8$ $D5$ 63 $3D$ 03 $A8$ 29 FE FC DF 23
Ronda 5	$A1$ 78 10 $4C$ 63 $4F$ $E8$ $D5$ $A8$ 29 $3D$ 03 FC DF 23 FE	$F1$ $C1$ $7C$ $5D$ 00 92 $C8$ $B5$ $6F$ $4C$ $8B$ $D5$ 55 EF 32 $0C$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	25 BD $B6$ $4C$ $D1$ 11 $3A$ $4C$ $A9$ $D1$ 33 $C0$ AD 68 $8E$ $B0$	$E1$ $E8$ 35 97 FB $C8$ $6C$ $4F$ 96 AE $D2$ FB $7C$ $9B$ BA 53
Ronda 6	$E1$ $E8$ 35 97 $4F$ FB $C8$ $6C$ $D2$ FB 96 AE $9B$ BA 53 $7C$	$E0$ $C8$ $D9$ 85 92 63 $B1$ $B8$ $7F$ 63 35 BE $E8$ $C0$ 50 01	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	$0F$ 60 $6F$ $5E$ $D6$ 31 $C0$ $B3$ DA 38 10 13 $A9$ BF $6B$ 01	52 85 $E3$ $F6$ $A4$ 11 CF 50 $C8$ $6A$ $2F$ $5E$ 94 28 $D7$ 07
Ronda 7	52 85 $E3$ $F6$ 50 $A4$ 11 CF $2F$ $5E$ $C8$ $6A$ 28 $D7$ 07 94	48 67 $4D$ $D6$ $6C$ $1D$ $E3$ $5F$ $4E$ $9D$ $B1$ 58 EE $0D$ 38 $E7$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	75 20 53 BB EC $0B$ $C0$ 25 09 63 CF $D0$ 93 33 $7C$ DC	AC EF 13 45 $C1$ $B5$ 23 73 $D6$ $5A$ CF 11 $B8$ $7B$ DF $B5$
Ronda 8	AC EF 13 45 73 $C1$ $B5$ 23 CF 11 $D6$ $5A$ $7B$ DF $B5$ $B8$	AA 61 82 68 $8F$ DD $D2$ 32 $5F$ $E3$ $4A$ 46 03 EF $D2$ $9A$	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	58 $1B$ DB $1B$ $4D$ $4B$ $E7$ $6B$ CA $5A$ CA $B0$ $F1$ AC $A8$ $E5$	49 45 $7F$ 77 DB 39 02 DE 87 53 $D2$ 96 $3B$ 89 $F1$ $1A$
Ronda 9	49 45 $7F$ 77 DE DB 39 02 $D2$ 96 87 53 89 $F1$ $1A$ $3B$	$A4$ 68 $6B$ 02 $9C$ $9F$ $5B$ $6A$ $7F$ 35 EA 50 $F2$ $2B$ 43 49	AC 19 28 57 77 FA $D1$ $5C$ 66 DC 29 00 $F3$ 21 41 $6E$	04 $E0$ 48 28 66 CB $F8$ 06 81 19 $D3$ 26 $E5$ $9A$ $7A$ $4C$	$D4$ $E0$ $B8$ $1E$ BF $B4$ 41 27 $5D$ 52 11 98 30 AE $F1$ $E5$
Ronda 10	$D4$ $E0$ $B8$ $1E$ 27 BF $B4$ 41 11 98 $5D$ 52 AE $F1$ $E5$ 30	19 $A0$ $9A$ $E9$ $3D$ $F4$ $C6$ $F8$ $E3$ $E2$ $8D$ 48 BE $2B$ $2A$ 08	$2B$ 28 AB 09 $7E$ AE $F7$ CF 15 $D2$ 15 $4F$ 16 $A6$ 88 $3C$	32 88 31 $E0$ 43 $5A$ 31 37 $F6$ 30 98 07 $A8$ $8D$ $A2$ 34	
Salida	32 88 31 $E0$ 43 $5A$ 31 37 $F6$ 30 98 07 $A8$ $8D$ $A2$ 34				

Cuadro 18: Descifrado de un bloque con AES.

Texto plano obtenido: 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34

Lo que muestra que el autómata celular tiene capacidad de cifrado igual a la del algoritmo AES-128.

La segunda prueba consiste en cifrar un bloque mediante AES-128 bits y descifrarlo con el autómata celular, de manera que la entrada y la salida deben ser exactamente las mismas, este proceso lo ilustra la imagen 71.

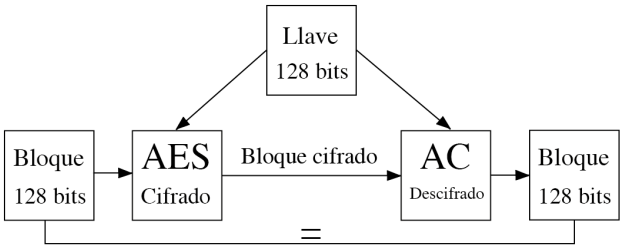

Figura 71: Cifrado de un bloque con AES y descifrado con AC.

Cifrando el bloque de entrada mediante AES y con la llave de entrada ya mencionadas

Ronda	SubBytes	ShiftRows	MixColumns	SubClave	AddRoundKey
Entrada	32 88 31 E0 43 5A 31 37 F6 30 98 07 A8 8D A2 34			2B 28 AB 09 7E AE F7 CF 15 D2 15 4F 16 A6 88 3C	19 A0 9A E9 3D F4 C6 F8 E3 E2 8D 48 BE 2B 2A 08
Ronda 1	D4 E0 B8 1E 27 BF B4 41 11 98 5D 52 AE F1 E5 30	D4 E0 B8 1E BF B4 41 27 5D 52 11 98 30 AE F1 E5	04 E0 48 28 66 CB F8 06 81 19 D3 26 E5 9A 7A 4C	A0 88 23 2A FA 54 A3 6C FE 2C 39 76 17 B1 39 05	A4 68 6B 02 9C 9F 5B 6A 7F 35 EA 50 F2 2B 43 49
Ronda 2	49 45 7F 77 DE DB 39 02 D2 96 87 53 89 F1 1A 3B	49 45 7F 77 DB 39 02 DE 87 53 D2 96 3B 89 F1 1A	58 1B DB 1B 4D 4B E7 6B CA 5A CA B0 F1 AC A8 E5	F2 7A 59 73 C2 96 35 59 95 B9 80 F6 F2 43 7A 7F	AA 61 82 68 8F DD D2 32 5F E3 4A 46 03 EF D2 9A
Ronda 3	AC EF 13 45 73 C1 B5 23 CF 11 D6 5A 7B DF B5 B8	AC EF 13 45 C1 B5 23 73 D6 5A CF 11 B8 7B DF B5	75 20 53 BB EC 0B C0 25 09 63 CF D0 93 33 7C DC	3D 47 1E 6D 80 16 23 7A 47 FE 7E 88 7D 3E 44 3B	48 67 4D D6 6C 1D E3 5F 4E 9D B1 58 EE 0D 38 E7
Ronda 4	52 85 E3 F6 50 A4 11 CF 2F 5E C8 6A 28 D7 07 94	52 85 E3 F6 A4 11 CF 50 C8 6A 2F 5E 94 28 D7 07	0F 60 6F 5E D6 31 C0 B3 DA 38 10 13 A9 BF 6B 01	EF A8 B6 DB 44 52 71 0B A5 5B 25 AD 41 7F 3B 00	E0 C8 D9 85 92 63 B1 B8 7F 63 35 BE E8 C0 50 01
Ronda 5	E1 E8 35 97 4F FB C8 6C D2 FB 96 AE 9B BA 53 7C	E1 E8 35 97 FB C8 6C 4F 96 AE D2 FB 7C 9B BA 53	25 BD B6 4C D1 11 3A 4C A9 D1 33 C0 AD 68 8E B0	D4 7C CA 11 D1 83 F2 F9 C6 9D B8 15 F8 87 BC BC	F1 C1 7C 5D 00 92 C8 B5 6F 4C 8B D5 55 EF 32 0C
Ronda 6	A1 78 10 4C 63 4F E8 D5 A8 29 3D 03 FC DF 23 FE	A1 78 10 4C 4F E8 D5 63 3D 03 A8 29 FE FC DF 23	4B 2C 33 37 86 4A 9D D2 8D 89 F4 18 6D 80 E8 D8	6D 11 DB CA 88 0B F9 00 A3 3E 86 93 7A FD 41 FD	26 3D E8 FD 0E 41 64 D2 2E B7 72 8B 17 7D A9 25
Ronda 7	F7 27 9B 54 AB 83 43 B5 31 A9 40 3D F0 FF D3 3F	F7 27 9B 54 83 43 B5 AB 40 3D 31 A9 3F F0 FF D3	14 46 27 34 15 16 46 2A B5 15 56 D8 BF EC D7 43	4E 5F 84 4E 54 5F A6 A6 F7 C9 4F DC 0E F3 B2 4F	5A 19 A3 7A 41 49 E0 8C 42 DC 19 04 B1 1F 65 0C
Ronda 8	BE D4 0A DA 83 3B E1 64 2C 86 D4 F2 C8 C0 4D FE	BE D4 0A DA 3B E1 64 83 D4 F2 2C 86 FE C8 C0 4D	00 B1 54 FA 51 C8 76 1B 2F 89 6D 99 D1 FF CD EA	EA B5 31 7F D2 8D 2B 8D 73 BA F5 29 21 D2 60 2F	EA 04 65 85 83 45 5D 96 5C 33 98 B0 F0 2D AD C5
Ronda 9	87 F2 4D 97 EC 6E 4C 90 4A C3 46 E7 8C D8 95 A6	87 F2 4D 97 6E 4C 90 EC 46 E7 4A C3 A6 8C D8 95	47 40 A3 4C 37 D4 70 9F 94 E4 3A 42 ED A5 A6 BC	AC 19 28 57 77 FA D1 5C 66 DC 29 00 F3 21 41 6E	EB 59 8B 1B 40 2E A1 C3 F2 38 13 42 1E 84 E7 D2
Ronda 10	E9 CB 3D AF 09 31 32 2E 89 07 7D 2C 72 5F 94 B5	E9 CB 3D AF 31 32 2E 09 7D 2C 89 07 B5 72 5F 94		D0 C9 E1 B6 14 EE 3F 63 F9 25 0C 0C A8 89 C8 A6	
Salida	39 02 DC 19 25 DC 11 6A 84 09 85 0B 1D FB 97 32				

Cuadro 19: Cifrado de un bloque con AES

Cifra obtenida: 3 9 2 5 8 4 1 D 0 2 D C 0 9 F B D C 1 1 8 5 9 7 1 9 6 A 0 B 3 2

Se procede a descifrar con el autómata mezclando la cifra obtenida por AES y la décima

expansión de la llave de cifrado:

3 9 D 0 2 5 1 4 8 4 F 9 1 D A 8 2 C 9 D C E E 9 2 5 F B 8 9 D C E 1 1 1 3 F 8 5 C 9 7 C 8 1 9 B 6 6 A 6 3 B C 3 2 A 6

Codificando a bloque de estados mediante L_{Box} obtenemos la configuración inicial del autómata celular:

F 0 5 3 B 9 3 4 F 4 6 3 6 9 D 8 1 9 9 5 5 6 E 3 C 7 B 9 8 0 A E 5 6 1 1 0 4 8 E E A 3 3 F 5 B C 7 1 B 1 2 8 C 3 6 8 3 3 8 A 4 9

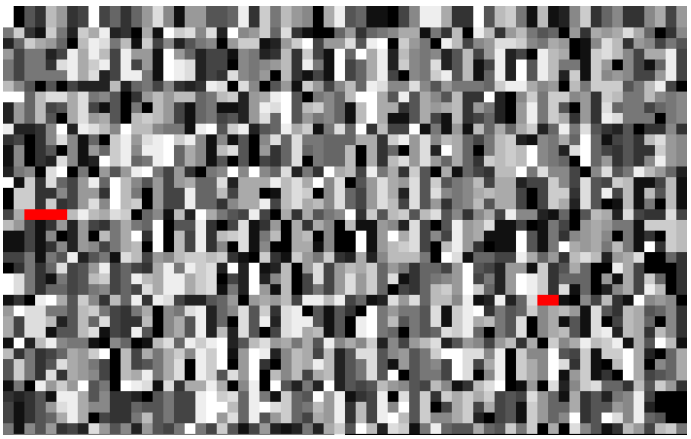

Figura 72: Proceso de descifrado de un bloque de 128 bits con el autómata celular

Al cabo de 41 evoluciones el autómata celular obtiene la cifra equivalente a AES aplicando las reglas caracterizadas, como se muestra en la siguiente figura 70, la configuración final obtenida en la lattice es:

8 A 8 1 B D A 9 F E 2 D D 8 B 1 4 F 5 6 E 2 4 0 6 5 6 3 E 6 3 5 2 F 8 5 2 F A C 5 9 8 E F 6 6 E 4 4 4 A 2 4 7 4 C 6 9 4 2 1 D F

Decodificando la configuración final del autómata celular mediante A_{Box} obtenemos una configuración que tiene incrustado el texto plano equivalente al descifrado AES en sus células pares:

3 2 1 6 4 3 9 D F 6 A B A 8 B 6 8 8 D C 5 A 4 C 3 0 F 9 8 D 3 C 3 1 6 D 3 1 7 A 9 8 3 F A 2 6 1 E 0 F 1 3 7 8 7 7 4 7 3 4 E

Conservando las células pares obtenemos:

3 2 4 3 F 6 A 8 8 8 5 A 3 0 8 D 3 1 3 1 9 8 A 2 E 0 3 7 0 7 3 4

Lo que muestra que el autómata celular tiene capacidad de descifrado igual a la del algoritmo AES-128.

5.4.2. Cifrado de un mensaje compuesto por múltiples bloques

Con fines demostrativos se ha desarrollado una aplicación que cifra texto plano ASCII, en la cual se puede especificar que entidad de cifrado se usará para procesar el archivo y así obtener un criptograma, ya sea el algoritmo AES ó el autómata celular. Sin embargo como

ambas entidades proporcionan procesos equivalentes de cifrado/descifrado ambas pueden ser usadas para procesar cualquier tipo de información digital como imágenes, documentos con extensiones especificas, tramas, etc. El objetivo de desarrollar esta aplicación es la de mostrar un ejemplo concreto del cifrado de información utilizando ambas implementaciones.
Se cifra el siguiente archivo de texto llamado AcercaDeAES.txt

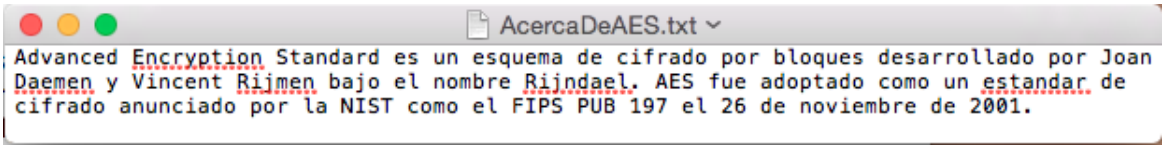

Figura 73: Archivo de texto llamado AcercaDeAES.txt

El archivo consta de 32 bloques, se procesa con el autómata celular en modo cifrado, aplicando a cada bloque el proceso de cifrado expuesto con anterioridad. La imagen 74 muestra la representación hexadecimal del archivo de entrada.

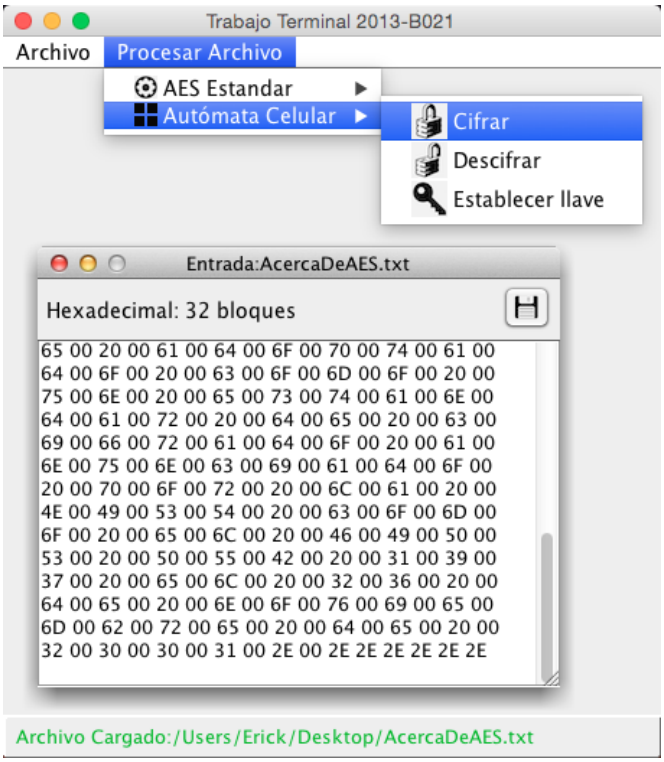

Figura 74: Representación hexadecimal del archivo de entrada.

Procesando el archivo mediante la opción de cifrado con autómata celular con la llave de cifrado:

Llave de cifrado = 2 B 7 E 1 5 1 6 2 8 A E D 2 A 6 A B F 7 1 5 8 8 0 9 C F 4 F 3 C

Al final se obtiene la siguiente gráfica que representa las lattices del autómata celular en la última transición que es donde se alcanza el cifrado, cada tira de pixeles es una lattice independiente de las demás, esto es por que no se implemento ningún modo de operación de encadenamiento por lo que modo sobre el que opera la aplicación es el denominado Electronic Code Book (ECB), en total hay 32 tiras, una por cada bloque.

Figura 75: Cifrado con autómata de un mensaje de texto que consta de 32 bloques.

Decodificando mediante A_{Box} se obtiene el hexadecimal del nuevo archivo de texto cifrado.

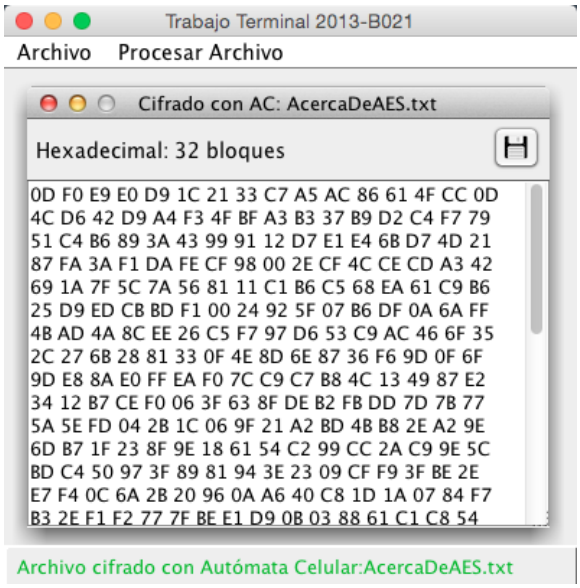

Figura 76: Representación hexadecimal de un mensaje de texto cifrado que consta de 32 bloques.

La representación en ASCII arroja el criptograma.

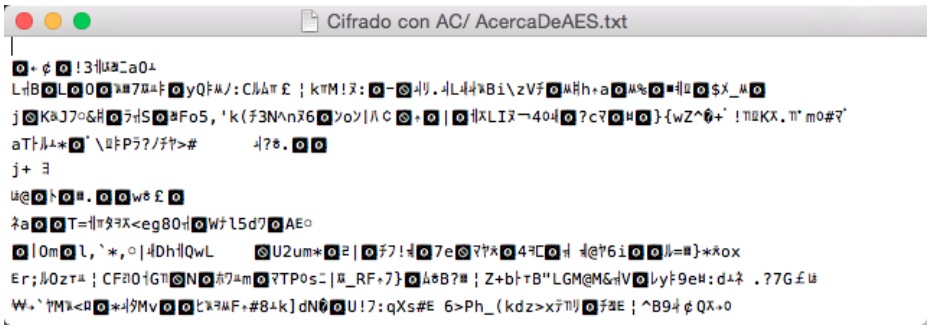

Figura 77: Criptograma generado por el autómata de un mensaje de texto cifrado.

Se procede a descifrar el archivo con el autómata célular y con el algoritmo AES de manera análoga a como se hizo para cifrarlo, estableciendo la misma llave.

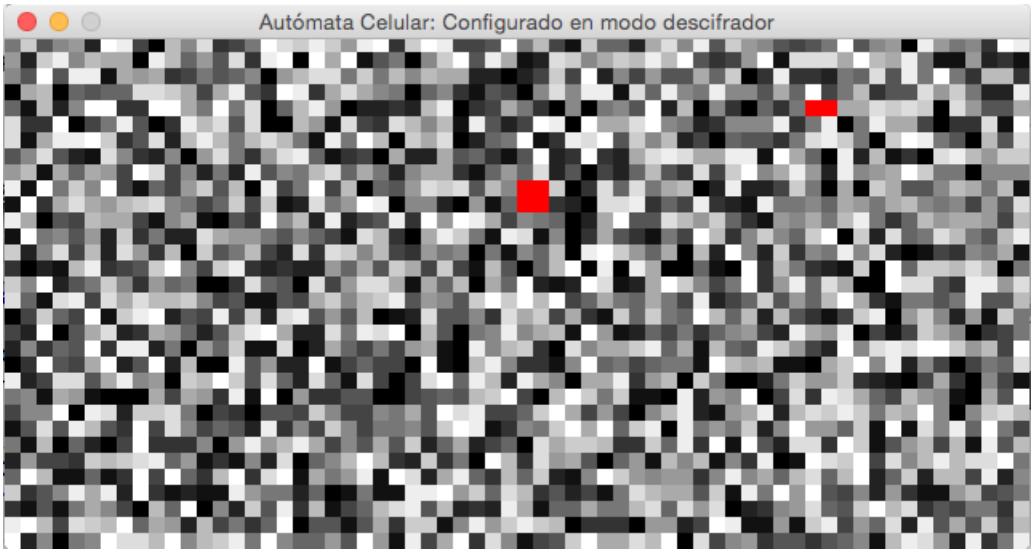

Figura 78: Descifrado con autómata de un mensaje de texto que consta de 32 bloques.

Descifrando con ambas entidades obtenemos las siguientes representaciones hexadecimales.

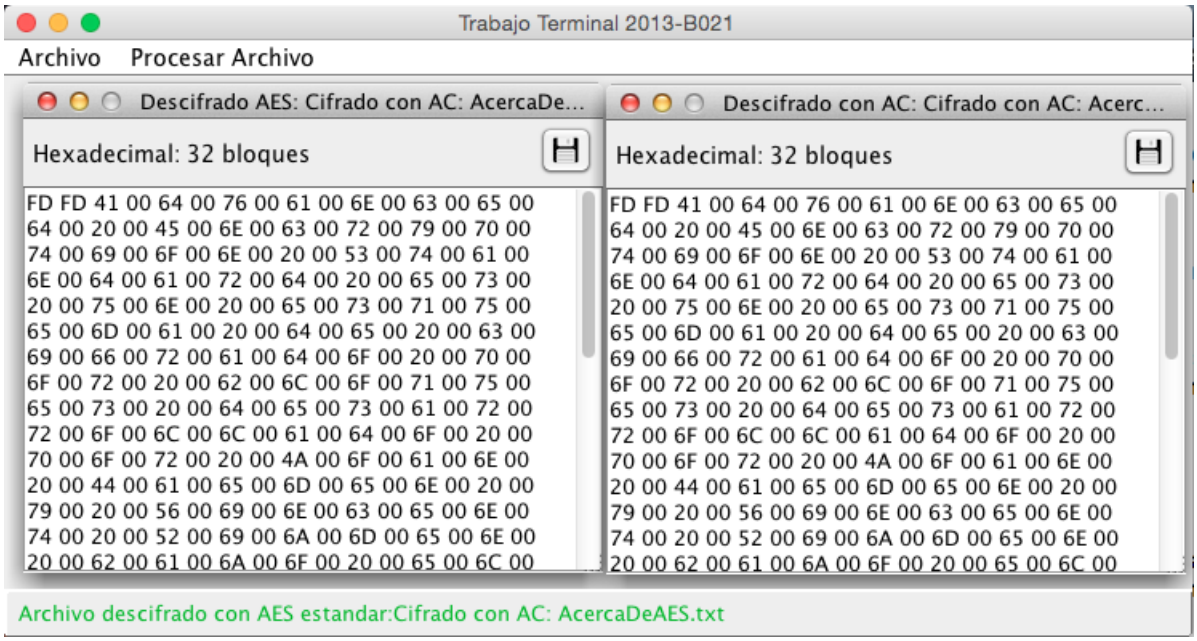

Figura 79: Representación hexadecimal del descifrado.

Notamos que ambas son iguales, es de esperarse puesto que las entidades poseen la misma capacidad de cálculo. Ambas representaciones arrojan el siguiente texto plano.

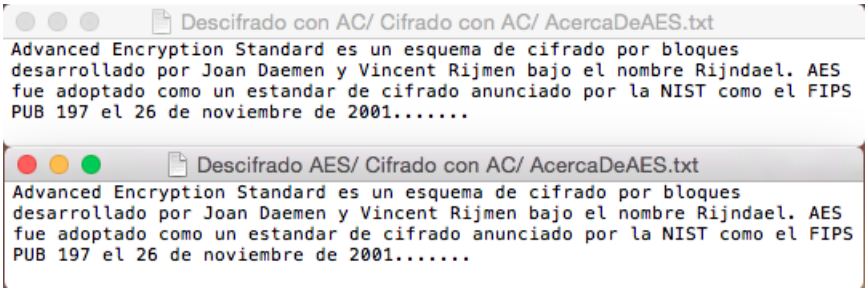

Figura 80: Textos planos generados por ambas entidades.

Observaciones Notamos que al final del archivo se han agregado una serie de puntos, esto se debe a que el mensaje original necesitaba de un relleno para poder a completar los 32 bloques, el relleno o padding se realiza agregando puntos (.) al final del texto plano de entrada.

Conclusiones y trabajo a futuro

6. Conclusiones

Se analizó el campo GF(2^8) tanto su grupo aditivo como el grupo multiplicativo, así como la transformación afín SBox que figura en el estándar AES, puesto que son las operaciones más básicas ya que transforman directamente las unidades de cifrado en dicho algoritmo, con el objetivo de caracterizar sus elementos y operaciones en términos de números enteros, para que estos puedan ser tratados como estados y funciones de transición de un autómata celular, haciendo que el autómata construido poseea capacidad de cálculo en aritmética del campo binario GF(2^8).

Derivado de dichas caracterizaciones se ha podido dar una enumeración distinta a la canónica y una codificación de los polinomios del campo así como de ciertas transformaciones del campo en forma de una representación matricial compacta a las que denominamos cajas, las cuales que poseen propiedades especiales que permiten hacer aritmética en campo finitos mediante sustituciones y sumas de números enteros. Además se ha logrado dar solución al problema de encontrar inversos multiplicativos prescindiendo del algoritmo extendido de Euclides para polinomios, así como la de expresar log(a+b) en términos de los estados del autómata, cabe mencionar que estos problemas no tienen soluciones triviales en el campo de la criptografía. Gracias a esto se ha podido diseñar un autómata con la capacidad de generar cifras codificadas y equivalentes al algoritmo AES en su versión de 128 bits. Como en criptografía no hay dos procedimientos ajenos que proveen las mismas cifras bajo las mismas entradas, quiere decir que existe un mapeo entre las cifras producidas por el autómata celular y las producidas por el algoritmo AES, y este mapeo esta dado precisamente por las funciones L_{BOX} y A_{BOX}, de esta manera se justifica que cifrar/descifrar con ambas entidades resulta indistinto puesto que las caracterizaciones los vuelven procesos equivalentes, lo cual implica que las construcciones y definiciones desarrolladas a lo largo del presente trabajo son una caracterización de un algoritmo criptográfico a través de un autómata celular.

Se mostró que el AC muestra una ventaja en tiempo en el descifrado respecto al AES clásico ya que las constantes de descifrado más operaciones por ser polinomios de grado mayor a los de cifrado, ya que en el cifrado son necesarias 32 productos polinomiales netos con a lo más una reducción modular, mientras que en el descifrado son necesarias 64 productos polinomiales netos con 4 al menos reducciones modulaes.

6.1. Trabajo a futuro

Dentro de las dinámicas de la reglas se observan comportamientos periódicos que resultan interesantes pues existe una fuerte sospecha que estos son generados cuando en la lattice es inicializada con elementos que forman subgrupos cilicios, además algunas matrices presentan ciertos patrones, lo que hace pensar que es posible realizar un ampliación de las cajas a campos con grados de extensión más altos, como GF(2^9) ó GF(2^{10}); es por ello que se propone como trabajo a futuro extender el cifrado a versiones de 192 y 256 bits, realizar un análisis más extenso y riguroso sobre el comportamiento de la dinámica de cada regla así como de la numeración de los polinomios, ya que es posible realizar adaptaciones del autómata celular a otras aplicaciones de los campos binarios, como la construcción de compresores de información, códigos lineales detectores de errores, generadores pseudoaleatorios y diversas aplicaciones de la teoría de información basados en autómatas celulares de manera similar a como se hizo con el algoritmo criptográfico.
Finalmente se desarrollo una aplicación del autómata al cifrado de texto plano ASCII, esto con el fin de mostrar un ejemplo más concreto al cifrado de información, sin embargo debe quedar claro que el autómata es capaz de cifrar cualquier instancia de información digital, por lo que

también se propone extender el uso de la caracterización oara el cifrado de información de distinta naturaleza al texto plano como imágenes, flujos, etc.

6.2. Glosario de terminología

- **Autómata Celular (AC):** sistema dinámico discreto que evoluciona en iteraciones a través de una regla determinística, tal como un sistema dinámico; las variables del sistema cambian como una función de sus valores actuales.
- **Advanced Encryption Standard (AES):** es un algoritmo de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos.
- **Anillo:** sea R un conjunto no vacío, con dos operaciones binarias $*$ y $\circ$, se dice que R es un anillo (R,$*$,$\circ$), si $*$ y $\circ$ cumplen los axiomas de anillo.
- **Bases polinomiales:** una base polinomial es una representación del campo GF(2^m) que esta compuesto por un conjunto de polinomios irreducibles sobre GF(2) con grado menor a m, de tal forma que cualquier elemento del campo es combinación lineal de polinomios irreducibles $f(x)$.
- **Bit:** dígito binario. Unidad mínima de almacenamiento de la información cuyo valor puede ser 0 ó 1; o bien verdadero o falso.
- **Byte:** conjunto de 8 bits el cual suele representar un valor asignado a un carácter.
- **Campo finito GF(2^m):** un campo de Galois binario GF(2^m) es un campo finito de característica 2 y extensión m.
- **Campo finito:** un campo finito consiste de un conjunto finito de elementos F sobre el cual se definen un par de operaciones binarias la suma $+$ y el producto $\bullet$.
- **Campos de Galois:** existe un campo de orden q si y solo q es potencia de un número primo p, es decir $q = p^m$ con $m \in \mathbb{N}$ dicho campo se denomina campo de Galois denotado GF(p^m) con característica q y grado de extensión m.
- **Célula:**es la unidad fundamental de la cual se compone una lattice.
- **Cifra: conjunto de los posibles mensajes cifrados.**
- **Cifrador de bloques:** es una unidad de cifrado de clave privada que opera en grupos de bits de longitud fija, llamados bloques, aplicándoles una transformación invariante.
- **Clave ó llave:** es una pieza de información que controla la operación de un algoritmo de criptografía. Por lo general, la clave es una secuencia de números o letras mediante la cual, se especifica la transformación del texto plano en texto cifrado, o viceversa.
- **Criptoanálisis:** es el estudio de técnicas matemáticas que se oponen a las técnicas criptográficas, pretende comprometer los servicios de seguridad de la información.
- **Criptografia asimétrica:** también llamada criptografía de clave pública, es el método criptográfico que usa un par de claves para el envío de mensajes. Las dos claves pertenecen a la misma persona que ha enviado el mensaje.
- **Criptografía simétrica:** también llamada criptografía de clave privada, es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes.

- **Criptografía:** ciencia que se dedica al estudio de la escritura secreta, es decir, estudia los mensajes que sometidos a un proceso de transformación, se convierten en difíciles o imposibles de leer.
- **Criptograma:** es un fragmento de mensaje cifrado cuyo significado resulta ininteligible hasta que es descifrado.
- **Curvas elípticas:**es una variante de la criptografía asimétrica o de clave pública basada en las matemáticas de las curvas elípticas.
- **Data Encryption Standard (DES):** es un método para cifrar información, escogido como un estándar FIPS en los Estados Unidos en 1976, y cuyo uso se ha propagado ampliamente por todo el mundo.
- **Esteganografía:** se ocupa de esconder mensajes con información privada, enviados por un canal inseguro, de forma que el mensaje sea desapercibido. Regularmente el mensaje se oculta dentro de datos con formatos de vídeo, imágenes, audio o mensajes de texto.
- **Estegoanálisis:** es la contraparte de la estenografía, la cual dedica su atención a de detectar mensajes ocultos con técnicas esteganográficas, para así reducir o eliminar la seguridad que aporta la esteganografía.
- **Función de transición:** es la entidad encargada de aplicar una función de transición a una configuración de vecindad proporcionada por la lattice.
- **Grupo:** sea G un conjunto no vació y la operación binaria $*$ sobre G, se dice que G es grupo (G,$*$) si $*$ cumple los axiomas de grupo.
- **Interface:** conexión entre dos componentes de hardware, entre dos aplicaciones o entre un usuario y una aplicación.
- **Latice:** ó espacio celular es una retícula regular L tal que si $d \in \mathbb{N}$ entonces $L = \{c|\mathbb{Z}^d\}$, para un lattice L de dimensión d.
- **Mensaje:** es lo que se denomina texto claro, o plaintext que pueden ser enviados a través de un canal de comunicación.
- **Secure And Fast Encription Routine (Safer):** es un algoritmo de cifrado por bloques no propietario. Está orientado a bytes y emplea un tamaño de bloque de 64 bits y claves de 64 (SAFER K-64) o 128 bits (SAFER K-128). Tiene un número variable de rotaciones, pero es recomendable emplear como mínimo 6.
- **Sistema dinámico discreto:** es un cuyo estado evoluciona con el tiempo. El comportamiento en dicho estado se puede caracterizar determinando los límites del sistema, los elementos y sus relaciones.
- **Transformación affín:** consiste en una transformación lineal seguida de una traslación, se define como $x \to \alpha x + \beta (mod\ n)$
- **Twofish:** es un método de criptografía simétrica con cifrado por bloques, El tamaño de bloque en Twofish es de 128 bits y el tamaño de clave puede llegar hasta 256 bits.

6.3. Glosario de notación

- $\bullet$ Producto en campo finito.

$<<$ Corrimiento aritmético hacia la izquierda.

$\oplus$ Xor, or exclusivo.

$\otimes$ Producto de dos polinomios (con grado menor 4) módulo $z^4 + 1$.

S_{box} Caja de sustitución.

$_{inv}S_{box}$ Caja de sustitución inversa.

$\delta(X_c, V)$ Función de transición en términos de la célula central y la vecindad V.

$\mathbb{F}_p$ Campo finito de característica p.

$\circ$ Puede significar operación binaria ó composición de funciones.

$p(x)$ Polinomio, elemento del campo de Galois.

GF(2^n) Campo de Galois de característica p y grado de extensión n.

$g(x)$ Elemento generatriz del grupo multiplicativo del campo de Galois.

L_{Box} Caja logarítmica.

A_{Box} Caja anti logarítmica.

E_{AC} Espacio de estados (conjunto finito de estados del autómata celular).

e Estado del autómata celular.

$\boxdot$ Regla del producto para el autómata celular.

LS_{Box} Caja de sustitución logarítmica.

$_{inv}LS_{Box}$ Caja inversa de sustitución anti logarítmica.

$\top$ Regla de la transformación affin para el autómata celular.

H Función H.

$\boxplus$ Regla de la suma para el autómata celular.

σ Permutación

σ^{-1} Permutación inversa.

Apéndice A: Pruebas y corridas

7. Apéndice A: Pruebas y corridas

Una vez que se ha implementado el sistema es necesario definir y diseñar el experimento que proveerá medidas reales acerca del comportamiento del algoritmo AES y el AC. El diseño del experimento consiste en definir un modelo de simulación validado y verificado, que se utiliza para predecir comportamientos de los sistemas en estudio mediante las corridas en una computadora y con cambios en sus variables de entrada o parámetros.

Cada corrida de simulación en una computadora es un experimento virtual. Por lo cual los cambios de valores de las variables de entrada deben organizarse con un diseño experimental para su correcta interpretación, ahorrando costos en tiempo y esfuerzo. Los resultados de las corridas según el diseño de experimentos seleccionado se analizan estadísticamente para detectar las fuentes de variabilidad: el error o las variables de entrada significativas que afectan la variable de salida o de respuesta.

Se denomina diseño experimental a pruebas en las cuales se inducen cambios deliberados en las variables de entrada de un proceso o sistema de manera que sea posible observar e identificar las causas de los cambios en la respuesta de salida.

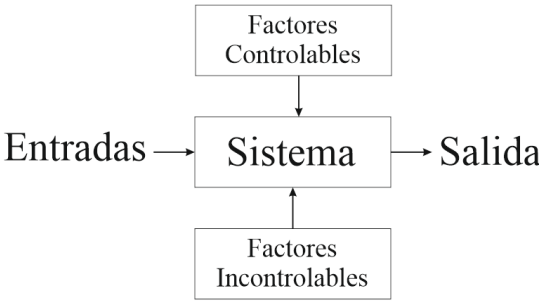

Figura 81: Parámetros al diseñar un experimento

Es prioritario establecer la pregunta o preguntas de investigación ya que es uno de los primeros recursos que se deben llevar a cabo cuando se diseña un experimento. La pregunta ó preguntas deben ser formuladas de manera precisa y clara, de tal manera que no exista ambigüedad respecto al tipo de respuesta esperada. Para el caso de estudio del presente trabajo terminal se plantean las siguientes preguntas que se pretenden contestar al termino del experimento.

1. ¿Cúal es el comportamiento promedio de ambas implementaciones?
2. ¿Cúal es la diferencia que hay entre los tiempos de ejecución de una implementación respecto a la otra?
3. ¿Cúal es la razón que hay entre los tiempos de ejecución de una implementación respecto a la otra?
4. ¿Cúal es el variación en los tiempos de ejecución de ambas implementaciónes?
5. ¿Cúal es la diferencia que hay entre la variación en el tiempo de ejecución de una implementación respecto a la otra?
6. ¿Cúal es la razón que hay entre la variación en el tiempo de ejecución de una implementación respecto a la otra?

7.1. Especificación del experimento

Un buen diseño de experimento permite hacer un análisis ordenado de los factores sospechados de influir en las respuestas o salidas de un sistema determinado. Para los fines del

trabajo terminal se tomará el modelo estadístico conocido como **Experimentos Factoriales**. El diseño de experimentos factoriales es ampliamente utilizado en diversas áreas de conocimiento que van desde la química hasta la mercadotecnia, en especial destaca su aplicación en el área computacional debido que permite contestar preguntas que se tienen acerca del rendimiento de un proceso ó bien del estudio de un modelo de cálculo, todo esto a través las corridas en computadora. Razones para escogerlo:

1. Permite determinar si uno o varios factores tienen efectos significativos sobre la variable de respuesta o si la diferencia observada en varias corridas de simulación es simplemente debida a las variaciones aleatorias causadas por errores y parámetros que no fueron controlados.

2. Permite separar los efectos de varios factores, de sus interacciones y del error que afecten al sistema.

3. Se obtiene la máxima información con el mínimo número de corridas, ahorrando trabajo y tiempo al obtener los resultados.

El presente diseño de experimentos tiene una base metodológica estadística para obtener conclusiones válidas y objetivas, ya que hay dos aspectos fundamentales: el diseño del experimento y el análisis estadístico de los datos. Además existen principios básicos a tener en cuenta para poder aplicar este experimento. El experimento asume que la asignación del entradas es aleatoria así como el orden de los ensayos y además estos deben tener la misma distribución de probabilidad. Los métodos estadísticos requieren que las observaciones (o errores) sean variables aleatorias independientes, para no trasladar otros errores que se producen en la secuencia de las observaciones, es por ello que concentrarse en el mejor o peor caso arrojara un análisis sesgado sobre el verdadero comportamiento de un sistema.

Terminología e Identificación de Variables

- *Variable de entrada*: Bloque de 128 bits (16 bytes).
- *Variable de respuesta*: Tiempo de ejecución del cifrado del bloque (Variable continua).
- *Factores:* Factores que intervienen directamente en el tiempo de ejecución del algoritmo:
    1. Valores del Bloque de entrada.
    2. Valores de la clave privada.
    3. Número de veces que se calcula el producto en campo finito GF(2^8) al cifrar/descifrar un bloque.
- *Niveles*: Son los valores que puede tomar un factor:

Factor	Mejor Caso	Peor Caso
Valores de la clave privada	$(00, .., 00)_{16}$	$(FF, .., FF)_{16}$
Valores del bloque de entrada	$(00, .., 00)_{16}$	$(FF, .., FF)_{16}$
Productos en campo finito GF(2^8)	576 productos •	576 productos •

Cuadro 20: Niveles de los factores que intervienen directamente en el tiempo de ejecución del algoritmo

- *Combinaciones:* Número de combinaciones entre niveles de los factores, 1 combinación de dos tomados dos al azar.

- *Corridas:* Pruebas o corridas de simulación.

$$(2\ casos\ para\ la\ clave)(2\ casos\ para\ el\ bloque)(576\ productos) = 2304\ corridas.$$

- *Réplicas:* Repetición de todos o algunos de los experimentos n veces.

1. 1 réplica para el cifrado AES.
2. 1 réplica para el descifrado AES.
3. 1 réplica para el cifrado con AC.
4. 1 réplica para el descifrado con AC.

Una ves obtenidos las muestras aleatorias se procederá a estimar los siguientes parámetros para poder responder las preguntas planteadas en este caso de estudio.

Replica	**Media**	**Varianza**
Cifrado AES	μ_{CAES}	σ^2_{CAES}
Cifrado AC	μ_{CAC}	σ^2_{CAC}
Descifrado AES	μ_{DAES}	σ^2_{DAES}
Descifrado AC	μ_{DAC}	σ^2_{DAC}

Cuadro 21: Parámetros a calcular

7.2. Herramientas de implementación

Se resolvió implementar con el paradigma de Programación Orientada a Objetos (POO) y el lenguaje de programación Java. Debido a que los programas estructurados tienen una debilidad para resolver programas complejos como lo es nuestro sistema. La primera las funciones tienen acceso ilimitado a los datos globales, segundo las funciones y datos, fundamentos del paradigma procedimental proporcionan un modelo pobre del mundo real. Java es un tipo de lenguaje de programación de propósito general, que aunque es usado para diferentes fines, en este caso sera empleado en el sistema, principalmente para cálculos matemáticos [22].

Una propiedad importante en POO es el polimorfismo que es aquella en la cual una operación que tiene el mismo nombre puede darse en diferentes clases y en cada uno de los casos la operación se ejecuta de manera diferente aun cuando en todos ellos tenga el mismo nombre. El polimorfismo es la propiedad de una operación de ser interpretada sólo por el objeto al que pertenece. En nuestro sistema aunque ambos casos cifren y tengan el mismo nombre lo realizan de manera diferente [22].

Para el caso del hardware se uso un equipo con la siguientes características.

Concepto	**Capacidad**
Marca	MacBookPro.
S.O.	MacOS X Yosemite.
Procesador	Dual core de 2.5 GHz Procesador Intel Core i5 con 3 MB de caché.
Memoria RAM	4 GB de memoria DDR3 de 1600 MHz
Disco duro	Disco duro de 500 GB.

Cuadro 22: Características del equipo utilizado para las corridas.

Para el caso del multiplicador polinomial para AES estandar, se usó el algoritmo conocido como algoritmo del la multiplicación en el campo finito GF(2^8) descrito en el marco teorico, (ver algoritmo 3).

7.3. Pruebas

Para cada una de las pruebas se introdujeron vectores de longitud con entradas aleatorias con un rango de 0 a 255. Se concluyo que se realizarían un total de 10 mil pruebas por cada una de las replicas tanto para el cifrado como para el descifrado con ambas implementaciones, de las cuales se obtuvieron los siguientes resultados:

Replica	**No. de pruebas**	**Media** ns	**Varianza** ns^2	**Desviación estándar** ns
Cifrado AES	10, 000	8,995.7	425,788,060.3160	20,634.633
Cifrado AC	10, 000	26,669.9	7,754,040,338.024	88,057.029
Descifrado AES	10, 000	26,234.2	179,558,293,779.738	423,743.193
Descifrado AC	10, 000	24,523.7	1,560,402,478.558	39,501.93

Cuadro 23: Parámetros calculados a partir de las muestras (unidades en nano segundos ns).

	Cifrado	**A favor de**	**Descifrado**	**A favor de**
Diferencia entre tiempos	17,674.2 ns	AES	1, 710.5 ns	AC
Razón entre tiempos	0.3372	AES	0.9347	AC
Diferencia entre la variación	67,422.396 ns	AC	384, 241.263 ns	AES
Razón entre la variación	0.2342	AC	0.0932	AES

Cuadro 24: Tiempos

7.4. Conclusiones del experimento

Como se puede apreciar en la tabla anterior, el tiempo promedio de cifrado es menor en el AES estándar, debido a que en la función de MixColumns, sus constantes de cifrado son polinomios con grados pequeños, por lo que el algoritmo del producto en el campo finito (ver algoritmo 3) realiza las operaciones en pocos pasos, para el descifrado el tiempo de ejecución de el autómata celular es menor que el tiempo promedio de AES estándar, debido a que los valores en la función InvMixColumns son polinomios con grados altos, por lo que el autómata puede operarlos haciendo menos pasos que el algoritmo del producto en el campo finito (ver algoritmo 3).

La variación del cifrado con AES es menor que la del AC, esto nos indica que no están muy separados del su tiempo promedio respecto al AC., cosa que ocurre en sentido contrario para el descifrado, que en esta ocasión la variación es menor en el AC.

7.4.1. Sobre otras pruebas

A continuación se presentan otras pruebas que se consideraron importantes, con el propósito de observar el comportamiento de ambos cifradores con entradas específicas. Las pruebas se realizaron con un solo bloque de 128 bits, con las características del hardware previamente mencionadas.

- **Prueba I:** en la primera prueba se introdujeron la llave y la entrada que se encuentra entre las pruebas que se le realizaron al algoritmo AES en el RFC.
  - **Llave:** 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
  - **Entrada:** 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34

Replica	No. de pruebas	Tiempos de ejecución ns	A favor de
Cifrado AES	1	217,165	AES
Cifrado AC	1	1,099,782,	
Descifrado AES	1	1,412,598	
Descifrado AC	1	351,816	AC

Cuadro 25: Parámetros calculados a partir de la prueba I (ns).

- **Prueba II:** en la segunda prueba se introdujo en la llave valores 00(hex) y en la entrada diversos valores.
  - **Llave:** 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - **Entrada:** 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34

Replica	No. de pruebas	Tiempos de ejecución ns	A favor de
Cifrado AES	1	148,197	AES
Cifrado AC	1	685,568	
Descifrado AES	1	669,147	
Descifrado AC	1	235,638	AC

Cuadro 26: Parámetros calculados a partir de la prueba II (ns).

- **Prueba III:** en la tercera prueba se introdujo una llave con diversos valores, pero en este caso la entrada se ha cargado con valores 00(hex)
  - **Llave:** 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
  - **Etrada:** 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Replica	No. de pruebas	Tiempos de ejecución ns	A favor de
Cifrado AES	1	960,616	
Cifrado AC	1	159,281	AC
Descifrado AES	1	1,071,866	
Descifrado AC	1	238,512	AC

Cuadro 27: Parámetros calculados a partir de la prueba III (ns).

- **Prueba IV:** en la la cuarta prueba se cargo la llave con valores FF(hex), así mismo con la entrada.
  - **Llave:** FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
  - **Entrada:** FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Replica	No. de pruebas	Tiempos de ejecución ns	A favor de
Cifrado AES	1	148,198	
Cifrado AC	1	684,747	AES
Descifrado AES	1	2,327,652	
Descifrado AC	1	678,589	AC

Cuadro 28: Parámetros calculados a partir de la prueba IV (ns).

- **Prueba V:** en la la quinta prueba se cargo la llave con valores 00(hex), así mismo con la entrada.
  - **Llave:** 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - **Entrada:** 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Replica	No. de pruebas	Tiempos de ejecución ns	A favor de
Cifrado AES	1	236,870	
Cifrado AC	1	673,252	AES
Descifrado AES	1	237,281	
Descifrado AC	1	354,369	AC

Cuadro 29: Parámetros calculados a partir de la prueba V (ns).

A continuación se presentan los histogramas del cifrado de ambas implementaciones, en las cuales se puede observar que el algoritmo AES estándar realiza el cifrado en menor tiempo, excepto en la prueba III los tiempos fueron tomados exclusivamente para las pruebas especificas antes mencionadas.

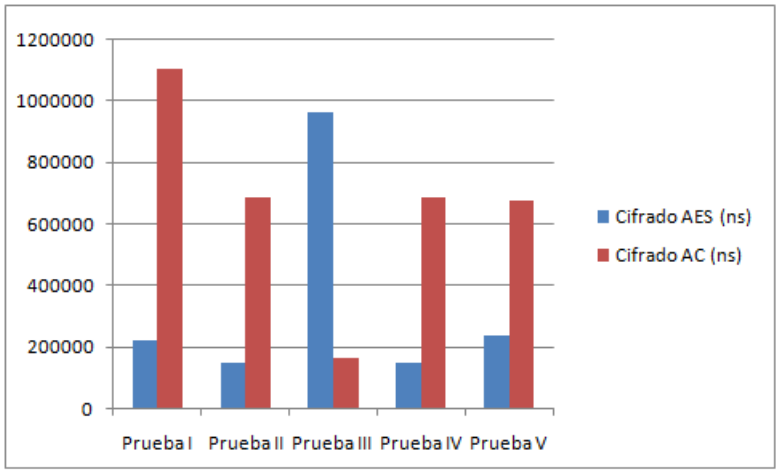

Figura 82: Histograma del cifrado de AES y AC

El siguiente histograma nos muestra gráficamente el descifrado de ambas implementaciones, en las cuales se puede observar que el algoritmo AC realiza el descifrado en menor tiempo, excepto en la prueba V, los tiempos fueron tomados exclusivamente para las pruebas especificas antes mencionadas.

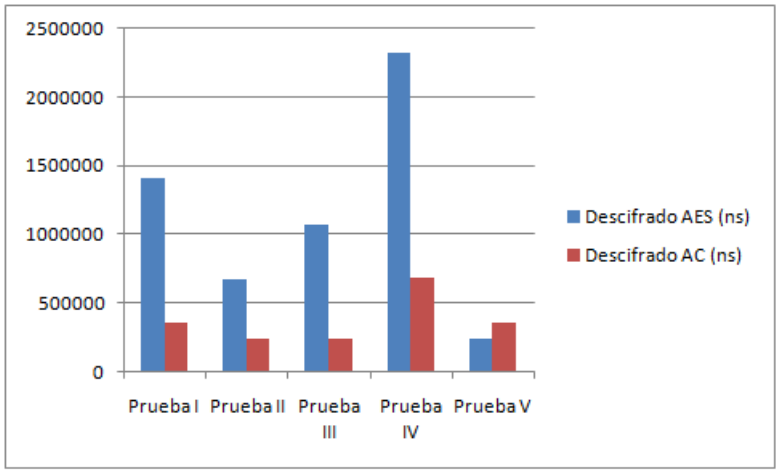

Figura 83: Histograma del cifrado de AES y AC

Se mostró que el AC muestra una ventaja en tiempo en el descifrado respecto al AES clásico ya que las constantes de descifrado más operaciones por ser polinomios de grado mayor a los de cifrado, ya que en el cifrado son necesarias 32 productos polinomiales netos con a lo más una reducción modular, mientras que en el descifrado son necesarias 64 productos polinomiales netos con 4 al menos reducciones modulaes.

Apéndice B: Manual de usuario

8. Apéndice B: Manual de Usuario

El sistema es una aplicación de escritorio que puede ser ejecutada desde cualquier computadora siempre y cuando tengan instalado java en el equipo.

La dirección URL para descargar java es:

http://www.java.com/es/download/

A continuación se muestra la pantalla principal del sistema. En la parte superior se encuentra el numero de TT, con dos de las opciones que pueden elegirse **Archivo** y **Procesar un bloque**, ademas de las opciones, minimizar, expandir y cerrar.

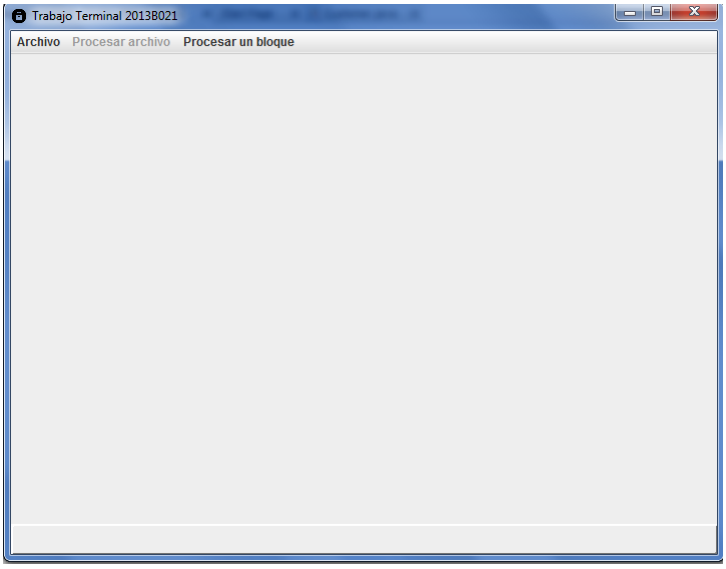

Figura 84: Pantalla principal del sistema

8.1. Procesar un bloque

El usuario debe seleccionar la opción de **Procesar un bloque**, el bloque puede ser procesado para cifrar/descifrar, cabe mencionar que es un bloque de 128 bits.

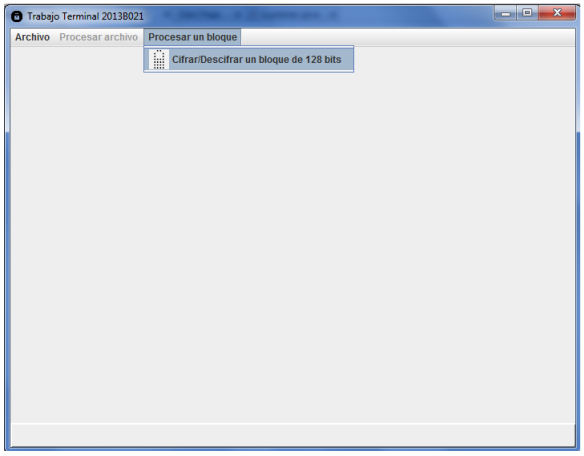

Figura 85: Pantalla procesar bloque

Una vez que el usuario ha seleccionado la opción **Procesar un bloque**, se abrirá inmediatamente una nueva ventana con los siguientes datos:

- **Bloque de 128 bits:** el usuario debe introducir un bloque a cifrar de 16 bytes escritos en hexadecimal.
- **Llave de 128 bits:** el usuario debe introducir una llave de 16 bytes escritos en hexadecimal
- **Cifra de 128 bits:** es la cifra que se obtiene después de procesar el bloque y la llave con la entidad o primitiva criptográfica.

Además se puede seleccionar entre las dos diferentes identidades criptográficas **Autómata Celular** y **Algoritmo AES** y entre dos diferentes primitivas **Cifrar** y **Descifrar**.

8.1.1. Cifrar bloque con autómata celular

Una vez que el usuario ha introducido el bloque a procesar, así como la llave, ambos de 128 bits, debe seleccionar la entidad criptográfica **Autómata Celular** y la primitiva **Cifrar**, y posteriormente dar click sobre el botón **Procesar Bloque**, se hará visible un gráfico con las evoluciones del autómata celular.

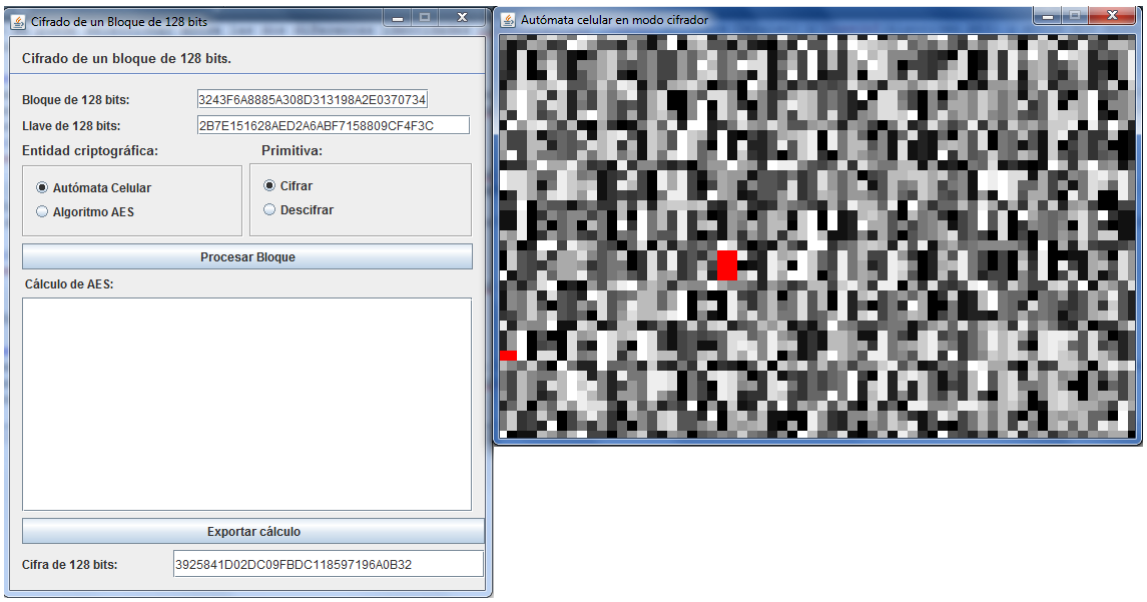

Figura 86: Menú para procesar un bloque y Lattice del autómata cifrador

Si se desea guardar el cálculo del cifrado, el usuario debe dar click en el botón **Exportar cálculo**, y seleccionar la ruta en la cual desea guardar el gráfico en formato .bmp, el cual se guardará con con el nombre **calculoCifradoAC**.

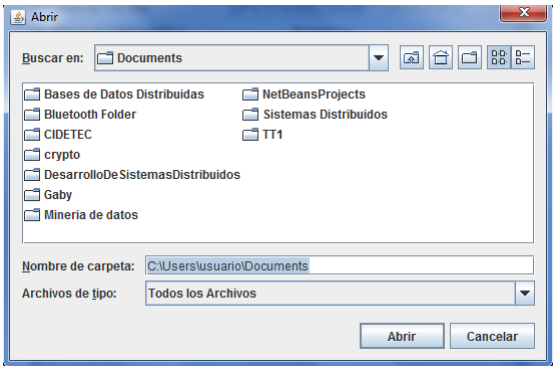

Figura 87: Ruta para exportar calculo del cifrado

8.1.2. Descifrar bloque con autómata celular

Una vez que el usuario ha introducido la cifra a procesar, así como la llave, ambos de 128 bits, debe seleccionar la entidad criptográfica **Autómata Celular** y la primitiva **Descifrar**, y posteriormente dar click sobre el botón **Procesar Bloque**, se hará visible un gráfico con las evoluciones del autómata celular.

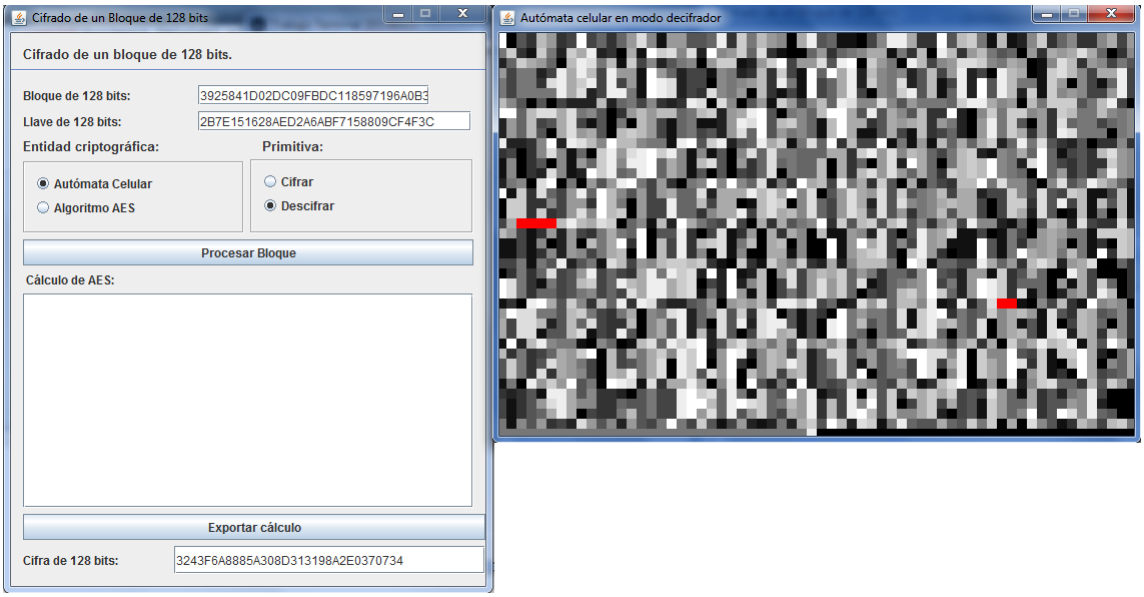

Figura 88: Menú para procesar un bloque y Lattice del autómata descifrador

Si se desea guardar el cálculo del descifrado, el usuario debe dar click en el botón **Exportar cálculo**, y seleccionar la ruta en la cual desea guardar el gráfico en formato .bmp, el cual se guardará con con el nombre **calculoDescifradoAC**.

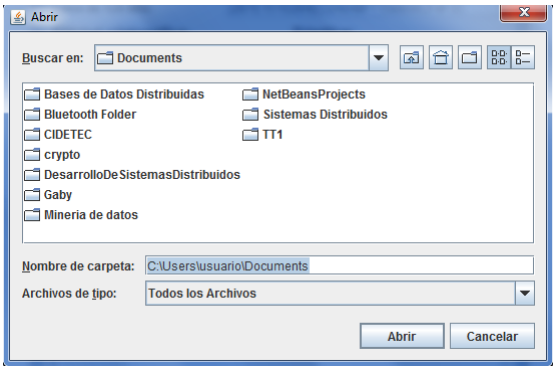

Figura 89: Ruta para exportar calculo del descifrado

8.1.3. Cifrar bloque con AES

Una vez que el usuario ha introducido el bloque a procesar, así como la llave, ambos de 128 bits, debe seleccionar la entidad criptográfica **Algoritmo AES** y la primitiva **Cifrar**, y posteriormente dar click sobre el botón **Procesar Bloque**, se hará visible en la pantalla los cálculos del algoritmo por función y por ronda.

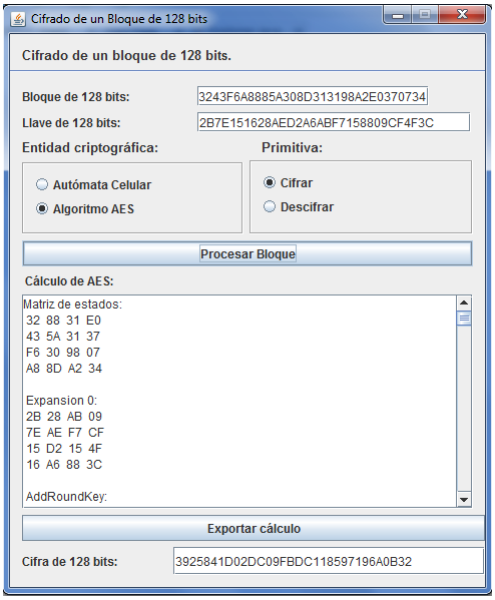

Figura 90: Menú para procesar un bloque y cálculos del algoritmo

Si se desea guardar el cálculo del cifrado, el usuario debe dar click en el botón **Exportar cálculo**, y seleccionar la ruta en la cual desea guardar el archivo en formato .txt, el cual se guardará con el nombre **CalculoCifradoAES**.

8.1.4. Descifrar bloque con AES

Una vez que el usuario ha introducido el bloque a procesar, así como la llave, ambos de 128 bits, debe seleccionar la entidad criptográfica **Algoritmo AES** y la primitiva **Descifrar**, y posteriormente dar click sobre el botón **Procesar Bloque**, se hará visible en la pantalla los cálculos del algoritmo por función y por ronda.

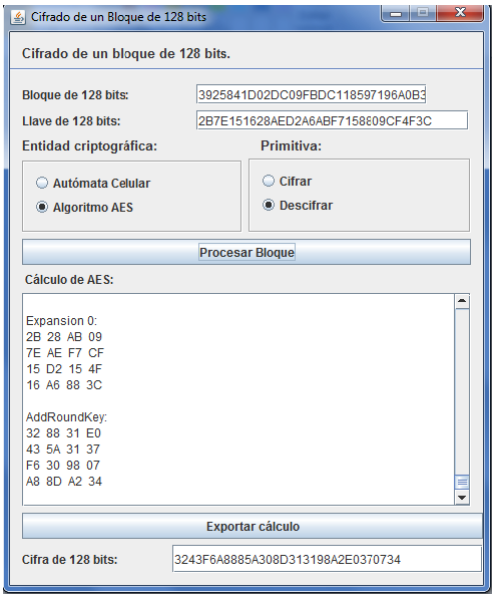

Figura 91: Menú para procesar un bloque y cálculos del algoritmo

Si se desea guardar el cálculo del cifrado, el usuario debe dar click en el botón **Exportar cálculo**, y seleccionar la ruta en la cual desea guardar el archivo en formato .txt, el cual se guardará con el nombre **CalculoDescifradoAES**.

8.2. Procesar archivo

El usuario desea procesar un archivo, debe seleccionar la opción **Archivo** y posteriormente elegir el recuadro **Abrir archivo de texto (.txt)** y elegir el archivo a procesar, es importarte que el usuario seleccione un archivo de texto en formato (.txt) ó **Salir** si desea cerrar la aplicación de escritorio.

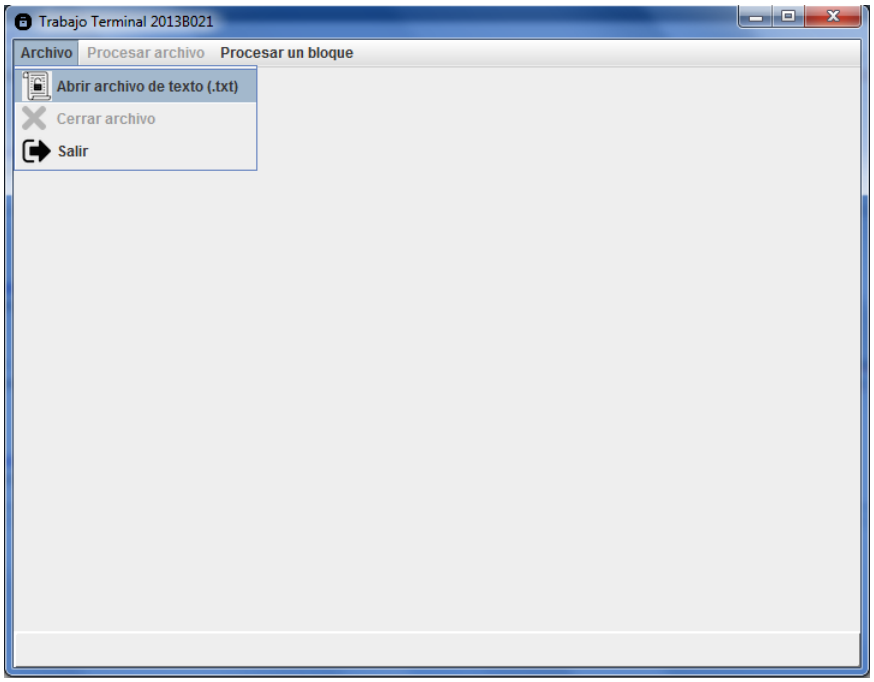

Figura 92: Pantalla abrir archivo

Una vez que se ha elegido el archivo a cifrar y se ha abierto, el sistema mostrará en pantalla

una ventana con título **Establecer Llave de cifrado**, el usuario debe introducir en formato hexadecimal una llave de 16 bytes y dar click en la opción **Establecer**, si el usuario no escribe la clave correctamente, aparecerán mensajes de error.

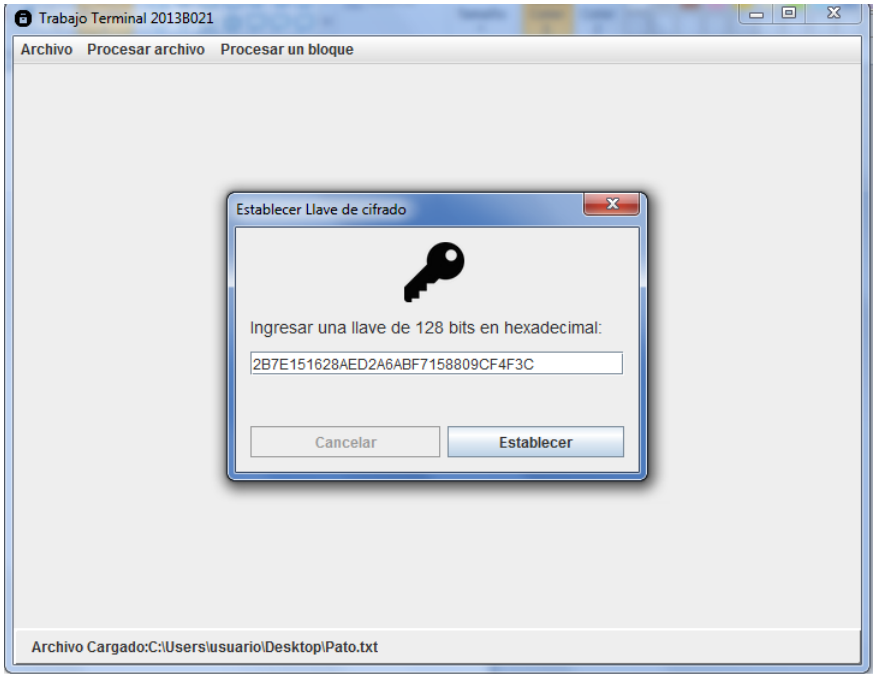

Figura 93: Pantalla establecer clave

El archivo se abrirá en el sistema en formato hexadecimal, con las opciones activas de **Procesar Archivo** así como la de cerrar y minimizar.

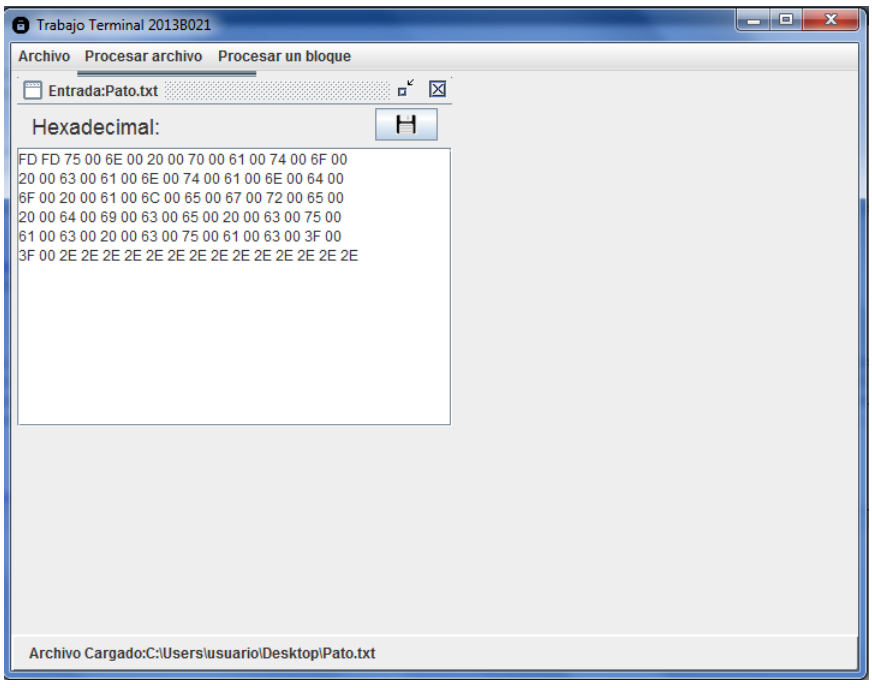

Figura 94: Pantalla archivo en formato hexadecimal

8.2.1. Cifrar archivo con AES

Una vez establecida la clave para el cifrado de el archivo, el usuario puede cambiar la clave, seleccionando la opción **Establecer Llave**.

Si el usuario desea procesar el archivo para cifrarlo debe elegir el recuadro **Procesar Archivo** y elegimos la opción **AES estándar**, seguido de la opción **Cifrar** esta opción mostrará en pantalla el archivo cifrado en formato hexadecimal, posteriormente el usuario deberá guardarlo, con el fin de poder procesar el archivo después.

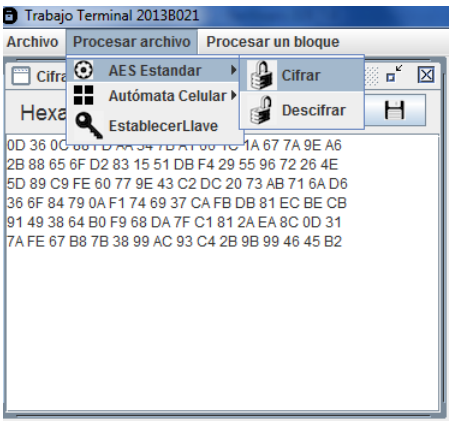

Figura 95: Pantalla cifrar AES estándar

El sistema guardará el archivo en formato (.txt), el archivo cifrado tomará la ruta en la que se encontraba el archivo que se abrió en primera instancia para procesar, en ese directorio se archivará con el nombre **Cifrado AES nombre del archivo**.

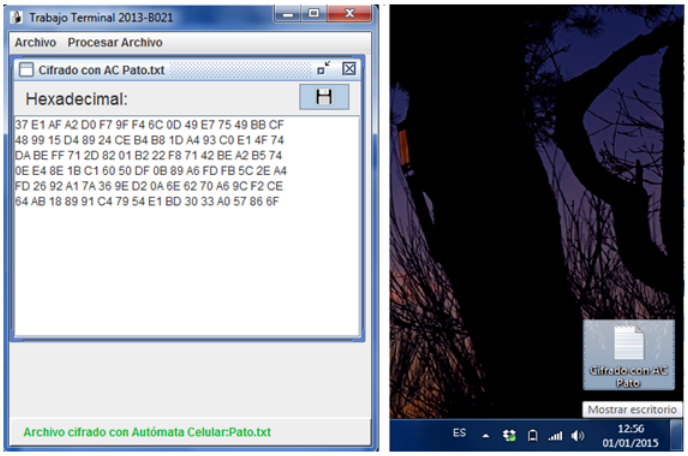

Figura 96: Pantalla guardar archivo cifrado

8.2.2. Descifrar archivo con AES

Una vez establecida la clave para el cifrado de el archivo, el usuario puede cambiar la clave, seleccionando la opción **Establecer Llave**.

Si el usuario desea descifrar un archivo, debe existir un archivo previamente cifrado y debe abrir el archivo cifrado en formato (.txt), la clave debe de ser la misma que se introdujo para cifrar el archivo, el usuario seleccionará la opción **Procesar Archivo**, y elegir la opción **AES**

estándar, seguido de la opción **descifrar** esta opción mostrará en pantalla el archivo descifrado en formato hexadecimal, posteriormente el usuario deberá guardarlo, con el fin de poder procesar el archivo posteriormente.

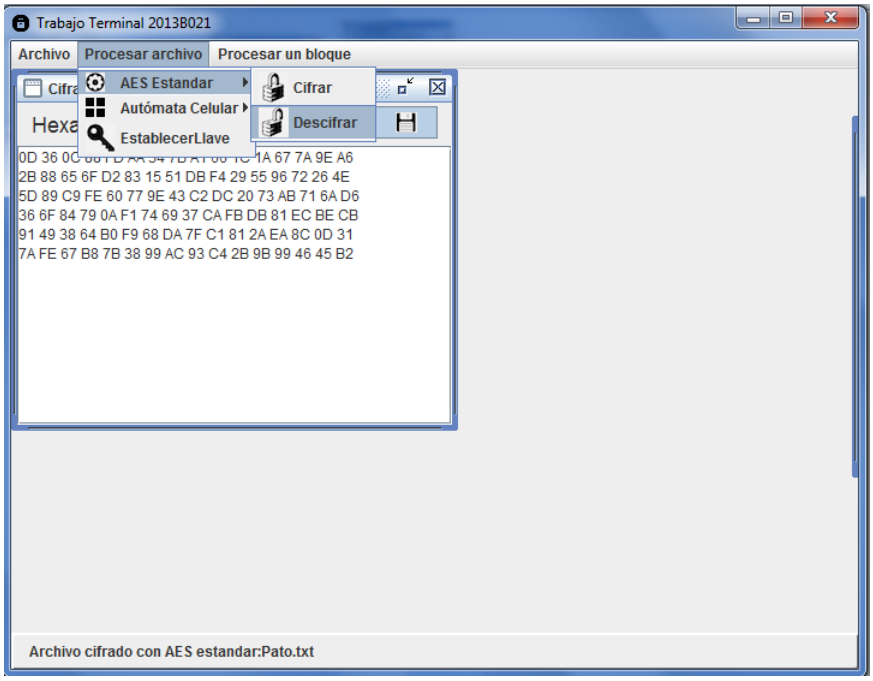

Figura 97: Pantalla descifrar AES estándar

El sistema guardará el archivo en formato (.txt), el archivo descifrado tomará la ruta en la que se encontraba el archivo que se abrió en primera instancia para procesar, en ese directorio se archivará con el nombre **Descifrado AES nombre del archivo**.

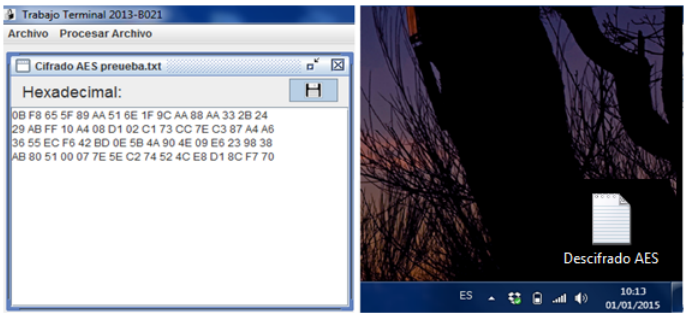

Figura 98: Pantalla guardar archivo descifrado

8.2.3. Cifrar archivo con autómata celular

Una vez establecida la clave para el cifrado de el archivo, el usuario puede cambiar la clave, seleccionando la opción **Establecer Llave**.

Si el usuario desea procesar el archivo para cifrarlo debe elegir el recuadro **Procesar Archivo** y elegimos la opción **Autómata Celular**, seguido de la opción **Cifrar** esta opción mostrará en pantalla el archivo cifrado en formato hexadecimal, posteriormente el usuario deberá guardarlo, con el fin de poder procesar el archivo después, además el sistema mostrará una pantalla mas con la evolución del autómata.

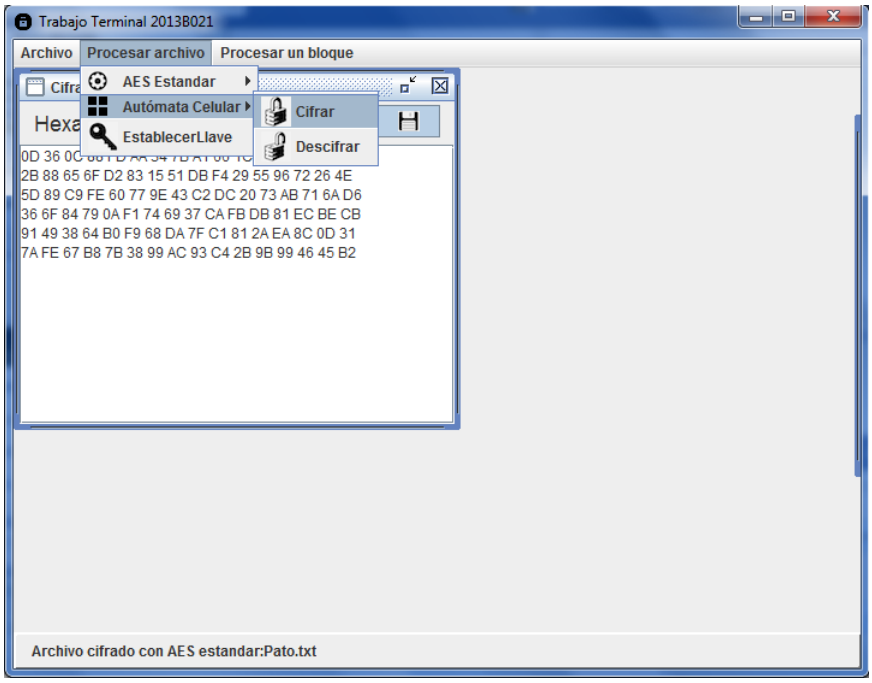

Figura 99: Pantalla cifrar Autómata celular

Figura 100: Pantalla evolución del Autómata Celular

El sistema guardará el archivo cifrado en formato (.txt), el archivo cifrado tomará la ruta en la que se encontraba el archivo que se abrió en primera instancia para procesar, en ese directorio se archivará con el nombre **Cifrado con AC**.

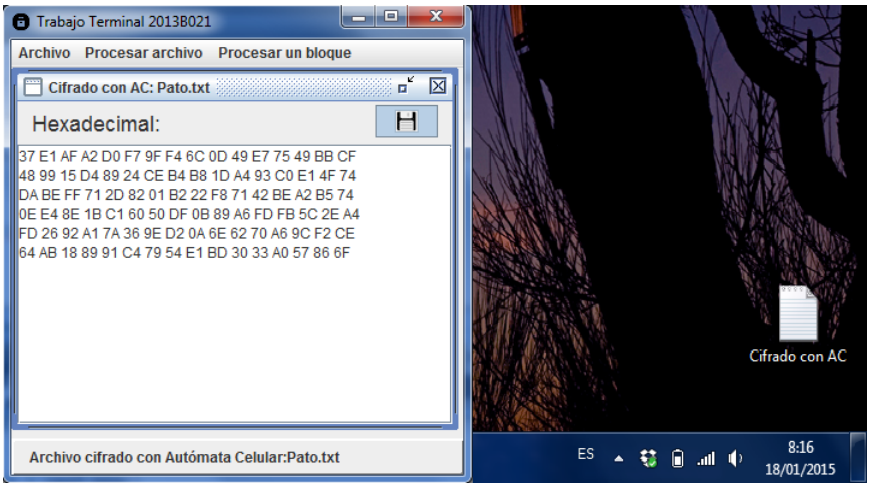

Figura 101: Pantalla guardar archivo cifrado

8.2.4. Descifrar archivo con autómata celular

Una vez establecida la clave para el cifrado de el archivo, el usuario puede cambiar la clave, seleccionando la opción **Establecer Llave**.

Si el usuario desea descifrar un archivo, debe existir un archivo previamente cifrado y debe abrir el archivo cifrado en formato (.txt), la clave debe de ser la misma que se introdujo para cifrar el archivo, el usuario seleccionará la opción **Procesar Archivo**, y elegir la opción **Autómata Celular**, seguido de la opción **descifrar** esta opción mostrará en pantalla el archivo descifrado en formato hexadecimal, posteriormente el usuario deberá guardarlo, con el fin de poder procesar el archivo después, además el sistema mostrará una pantalla mas con la evolución del autómata.

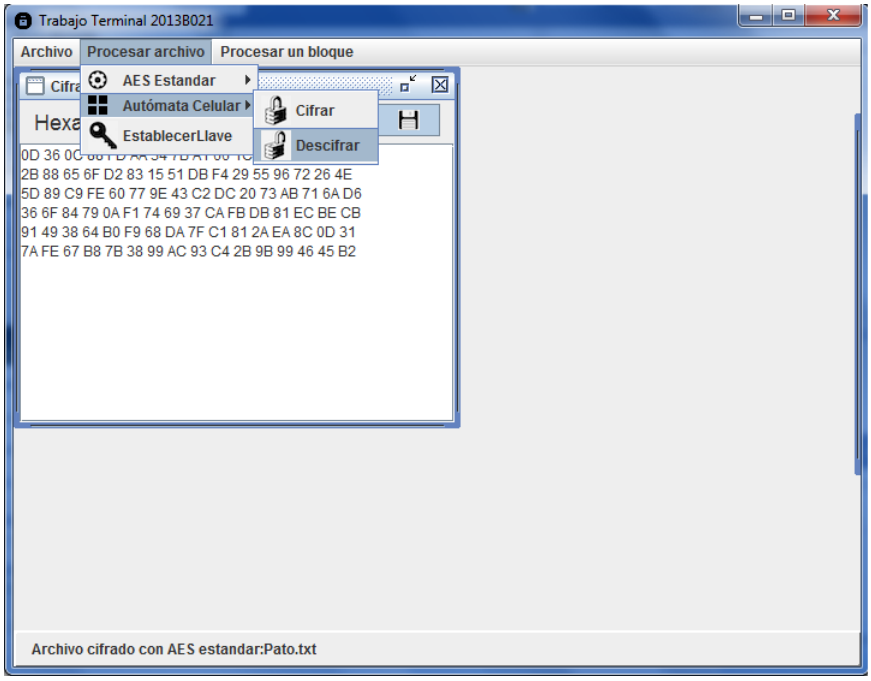

Figura 102: Pantalla descifrar autómata celular

Figura 103: Pantalla guardar archivo descifrado autómata celular

El sistema guardará el archivo descifrado en formato (.txt), el archivo tomará la ruta en la que se encontraba el archivo que se abrió en primera instancia para procesar, en ese directorio se archivará con el nombre **Descifrado con AC**.

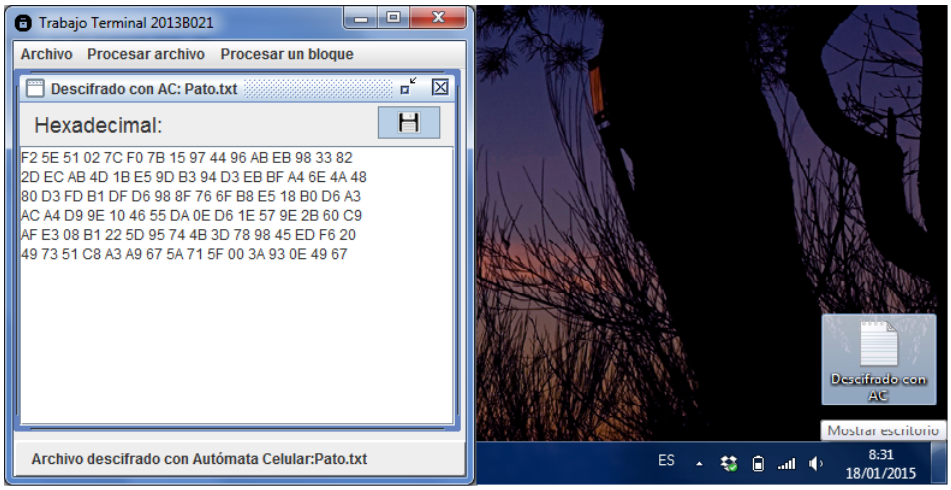

Figura 104: Pantalla guardar archivo descifrado

Referencias

[1] A. Joseph Raphael et. al. (2002, Mayo). *Cryptography and Steganography*. [Online]. Disponible:
http://www.ijcta.com/documents/volumes/vol2issue3/ijcta2011020338.pdf

[2] L. L. Manuel J. (2001,Junio). EPS [Online]. Disponible:
http://iie.fing.edu.uy/ense/asign/seguro/Criptografia.pdf

[3] *Handbook of applied Criptography*, RC PRESS, M. Alfred J. et. al., United States of America, 1997, pp 168.

[4] S. Alexander, *Introduction to cryptography with mathematical foundations and computer implementations*, CRC Press, United States of America, California,2010.

[5] L. O. Antonio, *Álgebra Superior I*, Las prensas de ciencias, Facultad de Ciencias U.N.A.M., México DF, 2012, pp 147-148.

[6] L. Rudolf, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, United Kingdom, 1986, pp 2-30.

[7] G. M. Mario Alberto, *"Construcción de operadores básicos sobre campos finitos GF(2^m)"*, Dr. en C. Tesis, Depto. Comp., CINVESTAV, México D.F.,2004. pp 19-39.

[8] T. R. Horacio, "Sobre algunas aplicaciones de los campos de Galois", Departamento de Matemáticas U.A.M.-I., México, DF, Miscelánea Matemática 53, 2011.

[9] L. H. Rotger et. al., "Criptografía con curvas elípticas", Universidad Abierta de Cataluña, Madrid España, Pid 00200952, 2009.

[10] J. M. Genaro, *"Grados de Reversibilidad en Autómatas Celulares Lineales"*, Lic. Tesis, Escuela Nacional de Estudios Profesionales Acatlán., U.N.A.M., México D.F., 1998.

[11] C. G. Abdiel, *"Máquina celular de computación basada en mosaicos regla 110"*, Dr. Tesis, Depto. Comp., CINVESTAV, México D.F., 2005. pp 24-40.

[12] H. T. Iliac, *"Simulación de sistemas naturales usando autómatas celulares"*, M. C. Tesis, Depto. Comp., CIC, México D.F., 2009. pp 12-17.

[13] R. Z. René, *"Modelación de flujo de tránsito de autos utilizando autómatas celulares"*, M. C. Tesis, Depto. Comp., CINVESTAV, México D.F.,2002. pp 19-39.

[14] W. Stephen, *A new Kind of Science / Stephen Wolfram*, Wolfram Media, United Kingdom, 2002.

[15] Jun-Cheol Jeon, Kee-Young Yoo, "An Evolutionary Approach to the Design of Cellular Automata Architecture for Multiplication in Elliptic Curve Cryptography over Finite Fields", Department of Computer Engineeing, Kyungpook National University, Korea, Daegu, 702-701.

[16] K. Kundan et. al., "A Programmable Parallel Structure to perform Galois Field Exponentiation.", IEEE Computer Society, ICIT, pp 277-280.

[17] L. O. Antonio, *AES - Advanced Encryption Standard*, Depto. Ing. Elect., CINVESTAV, México DF, 2005, pp 20-30.

[18] J. Daem, V. Rijmen, *Advanced Encryption Estandard (AES)*, Federal Information, Processing Standards Publication 197,2001. pp 14-26.

[19] M. M. Alfonso, *Algoritmo Criptográfico Rijndael*, Septiembre, 2004. pp 14-26.

[20] K. August, "La cryptographie militaire", *Journal des sciences militaires*, Vol. IX, pp. 5–83, January 1883.

[21] *Ingenieria de Software, un enfoque práctico*, Roger S. P., Mc Graw Hill, S. R. Pressman & Associates, Inc. España, 2002, pp 130, 275.

[22] J. A. Luis, *Programación en C++ un enfoque práctico*, McGraw-Hill, Madrid, España, 2006, pp.1-14.

[23] G.M. Emmanuel, "Un algoritmo de encriptación basado en la composición de las reglas 30 y 86 del autómata celular elemental", en: *Sistemas Complejos como Modelos de Computación*, J.M. Genaro, Z. Héctor, S. S. Cristopher R., (Eds.), Luniver Press, United Kingdom, 2011, pp. 167-180.

[24] C.D. Nidia Asunción , *"Multiplicadores de arquitectura segmentada y su aplicación al cómputo de emparejamientos bilineales"*, M.C. Tesis, Depto. Comp., CINVESTAV, México D.F.,2009.

[25] M.R. Fernando Carlos , P. M. Alma Delia, *"Encriptación de imágenes mediante autómatas celulares"* Trabajo Terminal, Depto. Ing. en Sis. Comp., ESCOM-IPN, México, D.F., 2008.

[26] W. C. Laurence, "The data encryption standard" *Introduction to cryptography with coding theory.* Pearson Prentice Hall, Maryland, U.S., 2012, pp. 113

[27] S. Marcin, B. Pascal, "Block cipher based on reversible cellular automata", Fclty. of Sci., Luxenbourg, Kirchberg, Rep. L-1359, Nov., 2004.

[28] S. Marcin et. al. "Cellular automata computations and secret key cryptography", in Tech. Paper, copyright 2004 ©, Parallel computing, doi: 4.3.1.

[29] P. P. Sambhu et. al., "Encryption and decryption algorithm using two dimensional cellular automata rules in cryptography", Dept. of MCA, India, Orrisa, Rep. 752054, 2011.

[30] S-T M., Juan Carlos, *"Análisis dinámico y topológico de los autómatas celulares unidimensionales reversibles"*, Dr. C., Tesis, Depto. Ing. Elect., CINVESTAV, México D.F., 2009.

[31] B.F. Rogelio León, H.P. Anaid, *"Computación basada en reacción de partículas en un autómata celular hexagonal"*, Trabajo Terminal, Depto. Ing. en Sis. Comp., ESCOM-IPN, México, D.F., 2009.

[32] V. Z. Elena, O. S. Francisco, "Autómatas celulares elementales aplicados a la encriptación de datos", en: *Sistemas Complejos como Modelos de Computación*, J.M. Genaro, Z. Héctor, S. S. Cristopher R., (Eds.), Luniver Press, United Kingdom, 2011, pp. 181-183.

[33] E. H. John et. al., *Introducción a la teoría de autómatas, lenguajes y computación*, Pearson Education, Madrid, España, 2007, pp.1-3.

[34] D. Kelley, *Teoría de Autómatas y Lenguajes Formales*, Pearson Education, Madrid, España, 2005, pp.30-31.

[35] A. A. Jhames, *Automata Theory with Modern Applications*, Cambridge University Press, United Kingdom, 2006, pp 169.

[36] A. Manuel et. al., *Teoría de lenguajes, Gramáticas y Autómatas*, Publicaciones R.A.E.C., Madrid España, 1997, pp 168-168.

[37] Neptalí Romero, “Comentarios sobre la definición de Autómata Celular ”, Boletín de la Asociación Matemática Venezolana, Venezuela, Vol. X, No. 1, Nov., 2003.

[38] Genaro J. Martínez, et. al., “Wolfram’s Classification and Computation in Cellular Automata Classes III and IV”, en: *Irreducibility and Computational Equivalence: 10 Years After Wolfram’s A New Kind of Science*, Genaro J. Martínez, Juan C. Seck-Tuoh-Mora, Hector Zenil, (Eds.), Springer Verlag, (2013). pp. 237-259.

[39] Genaro J. Martínez, et. al., “Computing on rings”, en: *A computable Universe: Understanding and Exploring Nature as Computation*, Genaro J. Martínez, Andrew Adamatzky, Harold V. McIntosh, (Eds.), WIRED, April 3, (2013).

[40] S. Marcin et. al. “Cellular automata computations and secret key cryptography”, in Tech. Paper, copyright 2004 ©, Parallel computing, doi: 4.3.1.

[41] V. M. Harold, “Conway’s Life’, en: *Game of Life Cellular Autómata*, Andrew Adamatzky (Ed.), Springier, United Kingdom, 2010, pp 513-539

[42] R. Paul, “A Turing Machine In Conway’s Game Life”, en: *Collision Based Computing*, Andrew Adamatzky (Ed.), Springier, United Kingdom, 2002, pp. 20-28.

[43] *Applied Cryptography, Protocols, Algorithms and Source code in C*, Jhon Wiley and Sons, S. Bruce, United States of America, 1996, pp 333.

[44] W. Diffie, H. Martin E. “New Directions in Cryptography”, *Transactions on Inforormation Theory*, Vol It-22, No. 6, pp. 644-653, Luniver Press, United Kingdom, 2011, November, 1976.

[45] S. Ian, *Ingeniería de Software*, Pearson Education, Madrid, España, 2005, pp.62.

URL’s

[46] Repositorio IPN. *Campos Finitos*. [Online]. Disponible: http://www.repositoriodigital.ipn.mx/bitstream/handle/123456789/7845/DOC6.pdf

[47] R. W. Neal (2001). *The Laws of Cryptography*. [Online]. Disponible: http://www.cs.utsa.edu/ wagner/laws/FFM.html/ijcta2011020338.pdf

[48] J. M. Genaro (2006). *Introducción a la simulación de procesos con autómata celular*. [Online]. Disponible: http://2006.igem.org/wiki/images/1/1b/IntoCA.pdf

[49] U.C.V. (2001). *Computación Emergente III: Introducción a los Autómatas Celulares*. [Online]. Disponible: http://fisica.ciens.ucv.ve/etisc/2001/curso/

[50] W. Stephen. (2001). *Elementary Cellular Automaton*. [Online]. Disponible: http://mathworld.wolfram.com/ElementaryCellularAutomaton.html

[51] Sergio Talens-Oliag (2003). *Introducción a la Criptología*. [Online]. Disponible: http://www.uv.es/ sto/articulos/BEI-2003-04/criptologia.html

[52] Elizabeth Pérez Cortés ,(2005), *Análisis de Algoritmos*. [Online]. Disponible: http://docencia.izt.uam.mx/pece/pagina/academica/AA/indexa.html

[53] H.B. Enderton (2009, Marzo). *Alonzo Church: Life and Work* [Online]. Disponible: http://www.math.ucla.edu/ hbe/church.pdf

[54] C. P. Juan (2009, Enero). *Modelos de Computación II* [Online]. Disponible: http://decsai.ugr.es/ castro/MCII/MCII.html

[55] Genaro J. Martínez et. al. Journal of Cellular Automata 7(5-6), 393-430, (2013). *Computation and Universality: Class IV versus Class III Cellular Automata*. [Online]. Disponible: http://www.oldcitypublishing.com/JCA/JCAabstracts/JCA7.5-6abstracts/JCAv7n5-6p393-430Martinez.html

[56] Stanford Encyclopedia of Philosophy (Junio 2012). *Turing Machines*. [Online]. Disponible: http://plato.stanford.edu/entries/turing-machine/

[57] Genaro J. Martínez. Journal of Cellular Automata 8(3-4), 233-259, (2013). *A Note on Elementary Cellular Automata Classification*. [Online]. Disponible: http://www.oldcitypublishing.com/JCA/JCAabstracts/JCA8.3-4abstracts/JCAv8n3-4p233-259Martinez.html

[58] R. Paul (2001). *The GoL Turing Machine*. [Online]. Disponible: https://www.ics.uci.edu/ welling/teaching/271fall09/Turing-Machine-Life.pdf

[59] Oracle (2014). *¿Qué es Java?*. [Online]. Disponible: http://http://www.java.com/es/download/faq/whatis-java.xml

[60] Eduardo José Jaramillo Villegas, (2014, Abril), *Análisis y Diseño de Algoritmos*. [Online]. Disponible: http://www.virtual.unal.edu.co/cursos/sedes/ manizales/4060024/Lecciones/Capitulo %20II/rbasicas.htm