CT-875-551
DOU: 2015

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Criptoanálisis Lógico-algebraico a Cifrados de Flujo Basados en Autómatas

Tesis que presenta:

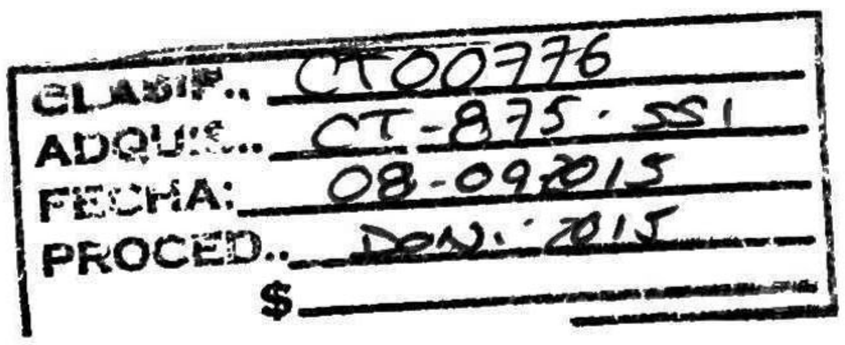
Luis Ricardo Peña Llamas

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr. Raúl Ernesto González Torres

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Enero de 2015.


CLASIF.. CT00776
ADQUIS.. CT-875-SSI
FECHA: 08-09-2015
PROCED.. Dep. 2015
$

Criptoanálisis Lógico-algebraico a Cifrados de Flujo Basados en Autómatas

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Luis Ricardo Peña Llamas

Ingeniero en Sistemas Computacionales
Instituto Tecnológico Superior de Zapopan 2007-2012

Director de Tesis

Dr. Raúl Ernesto González Torres

CINVESTAV del IPN Unidad Guadalajara, Enero de 2015.

Agradecimientos

A mis padres, Laura y Ricardo, que siempre me han ayudado para lograr mis metas. A mis hermanas, Ana Vanessa y Laura Esther por apoyarme y no dejarme rendir. A mi sobrina Paulette por regalarme tanta felicidad.

A mi asesor Dr. Raúl Ernesto González Torres por su valioso asesoramiento durante el proyecto de tesis y la formación adquirida.

A mis compañeros de generación por el apoyo recibido y experiencias adquiridas.

A mis amigos y compañeros del laboratorio de computación, por servir de guía y darme recomendaciones.

A CINVESTAV por otorgarme la oportunidad de realizar mi maestría.

A CONACYT, por el apoyo económico otorgado con la beca de manutención a través del Programa Nacional de Posgrados de Calidad.

Resumen

Presentamos una nueva forma de ataque dirigido hacia los criptosistemas que pueden ser modelados como un sistema de ecuaciones polinómicas. Combina los refinamientos de técnicas estándar de criptoanálisis algebraico y lógico, y emplea una nueva técnica, llamada recnf, que nos brinda mas información sobre el sistema original de su sistema lineal asociado. Con la ayuda de la técnica recnf, y mediante la imposición de restricciones a la FNC, el criptoanálisis se puede hacer más rápido sobre SAT-Solvers. Como casos de estudio tenemos dos criptosistemas: el criptosistema basado en autómatas finitos (FAPKC) y Bivium.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

En este capítulo, comenzamos exponiendo los antecedentes, que involucra información necesaria para el entendimiento del marco de este trabajo. Después, exponemos el problema a tratar. Posteriormente, la motivación a dicho problema. Luego, el objetivo de esta tesis. Seguidamente, mostramos el estado del arte. Por último, se describe la organización de la tesis.

1.1. Antecedentes

A través de los años se han hecho muchos estudios sobre la criptografía, uno de ellos es el de C. Shannon [Shannon, 1949], en donde probó que ningún cifrado es perfectamente seguro, a excepción del esquema *one-time-pad*. Sin embargo, la principal desventaja del esquema one-time-pad es que la clave necesita ser tan larga como el mensaje y este esquema es simétrico, por lo que es necesario que la clave sea conocida tanto por el emisor como por el receptor; entonces nos enfrentamos con el problema de cómo transmitir la clave y si encontramos la forma de transmitir la clave de forma segura, ¿por qué no mejor transmitir el mensaje? (ya que la clave tiene el mismo tamaño que el mensaje).

El problema de poder transmitir la clave de forma segura se le conoce como *el problema de la transmisión de claves*, el cual tiene como objetivo que ambas partes, tanto el emisor como el receptor, conozcan la clave privada de forma segura. Este problema fue investigado por Diffie y Hellman [Diffie y Hellman, 1976], en este trabajo muestran un algoritmo con el cual un par de usuarios comparten el mismo número, sin que alguna persona que esté observando el canal de comunicación entre ambas partes sea capaz de obtener el número que comparten.

Dos años después de que Diffie y Hellman encontraran solución al problema de transmisión de claves, Rivest, Shamir y Adleman [Rivest et al., 1978] propusieron un tipo de cifrado asimétrico basado en las ideas propuestas por Diffie y Hellman; este cifrado asimétrico tiene el nombre de RSA. La seguridad de RSA se basa en la factorización de dos números primos y para romper el criptosistema RSA es necesario conocer esta descomposición. Hasta ahora no se conoce un algoritmo eficiente para encontrar la descomposición de un número en dos números primos. Si se llegase a encontrar un algoritmo eficiente para dicha descomposición, el criptosistema RSA sería inseguro. Esto se debe a que una vez que se encuentra la descomposición se puede encontrar la clave privada fácilmente y una vez que ya conocemos la clave privada podemos descifrar los mensajes enviados.

Unos años antes de que RSA fuera propuesto, se creó el algoritmo de encriptación llamado DES (en el año 1977). Este algoritmo fue creado por un equipo de criptógrafos de IBM y fue modificado por la NSA (National Security Agency). DES cuenta con dos operaciones primitivas que, de acuerdo con Shannon, teniendo estas dos características se vuelve un algoritmo de encriptación fuerte. Estas dos operaciones son: la confusión y la difusión. La confusión es que no se revele la relación entre la clave y el texto cifrado y la difusión es que no se revele la relación entre el texto plano y el texto cifrado. Durante 23 años no fue conocido ningún ataque a este cifrado y no fue sino hasta el año 1991 cuando Biham y Shamir [Biham y Shamir, 1991] lograron atacar de forma exitosa a DES con un

nuevo enfoque llamado *criptoanálisis diferencial.*

En el año 2000 Massacci y Marraro [Massacci y Marraro, 2000] propusieron un nuevo enfoque para criptoanalizar a DES, llamado *criptoanálisis lógico*, el cual consiste en codificar las propiedades de los algoritmos de cifrado como problemas SAT y luego utilizar un sistema de prueba automática de teoremas y SAT-Solvers para obtener el texto plano o la clave privada del criptosistema.

Tres años después Courtois y Meier [Courtois y Meier, 2003] propusieron una nueva forma de criptoanalizar a los cifrados de flujo, llamada *criptoanálisis algebraico*, que consiste en modelar el criptosistema como una función y a cada paso de tiempo actualizar las variables de la función. Esto nos permite modelar el criptosistema como un sistema de ecuaciones polinómicas. Con ayuda de este tipo de criptoanálisis se probó que algunos cifrados de flujo, como TOYOCRYPT, son inseguros.

1.2. Planteamiento del problema

Se cree que resolver un sistema de ecuaciones polinómicas aleatorias es difícil (problema NP), de hecho Bard [Bard, 2009] dice:

Si se encuentra un algoritmo que resuelva un sistema de ecuaciones polinómicas en tiempo polinomial, entonces $P = NP$.

El problema de resolver un sistema de ecuaciones polinómicas sobre un campo finito es NP-difícil. En particular, cuando el grado de las ecuaciones es exactamente dos, se le conoce el problema MQ o MQP (por sus siglas en inglés, *Multivariate Quadratic Problem*); cuando el grado es al menos dos se le conoce como el problema MP o MPP (por sus siglas en inglés, *Multivariate Polynomials Problem*).

Los corolarios 79 y 80 en [Bard, 2009] nos dicen que los problemas MP y MQ nunca tendrán un algoritmo en tiempo polinomial que los resuelva, a menos que P = NP.

1.3. Motivación

La seguridad de los sistemas informáticos ha tomado una gran importancia, ya que cada vez más personas comparten información confidencial a través de estos medios y desean que está permanezca confidencial, es decir, que su información no sea conocida por personas no autorizadas. Debido a que no conocemos una forma de probar si los criptosistemas son seguros, lo único que podemos hacer para probar su seguridad es atacarlos y tratar de romperlos para poder observar las debilidades con las que cuentan y con esto encontrar una manera de reforzar el criptosistema y que la debilidad encontrada no sea una de sus debilidades. En caso de que no se pueda encontrar una manera de reforzar el criptosistema, hacer público que esté no es seguro debido a la vulnerabilidad encontrada.

1.4. Objetivo

El objetivo de esta tesis es proponer una nueva técnica de criptoanálisis aplicable a sistemas de cifrado basados en autómatas finitos. Utilizamos como base dos tipos de criptoanálisis: (1) el criptoanálisis algebraico y (2) el criptoanálisis lógico; utilizamos estas formas de criptoanálisis con el fin de obtener un sistema de ecuaciones polinómicas, de acuerdo a las especificaciones del criptosistema y después convertir el sistema de ecuaciones en una fórmula proposicional en *forma normal conjuntiva* (*FNC*), que también es un problema NP-difícil. Una vez obtenida la forma normal conjuntiva agregamos algunas restricciones, las cuales brindan cierta información adicional sobre el sistema de ecuaciones polinómicas original. Por

último utilizamos un SAT-Solver para obtener la clave privada.

1.5. Estado del arte

En esta sección damos una breve explicación de los avances más recientes en investigación que están más cercanos a nuestra propuesta. Damos una breve explicación de ellos. Para comprender un poco mejor las propuestas invitamos al lector a leer las referencias mostradas en las secciones correspondientes.

1.5.1. Criptoanálisis a Bivium utilizando SAT-Solvers de Eibach et al.

En el artículo presentado por Eibach et al. [Eibach et al., 2008] se muestran resultados experimentales de varios tipos de criptoanálisis efectuados a Bivium. Los ataques basados en SAT-Solvers son los más rápidos, con un tiempo de ejecución esperado de 1.64×10^{13} segundos; los otros ataques comparados son: búsqueda exhaustiva, ataques basados en BDDs, un enfoque con teoría de grafos y ataque basado en bases de Gröbner.

El ataque utilizando SAT-Solvers consiste en tres pasos: (1) obtener un sistema de ecuaciones, el cual describe el estado interno de Bivium, (2) convertir las ecuaciones a su forma normal conjuntiva, (3) intentar adivinar ciertas variables.

Las diferentes comparativas que muestra este articulo son:

- Los tiempos que tardan los diferentes SAT-Solvers como: Rsat, MiniSat, Picosat, SatElite y Zchaff.

- Las diferentes estrategias para adivinar variables como: las últimas 48 variables, las últimas variables de los dos diferentes registros, entre otras.

  Los cuatro diferentes criptoanálisis utilizados como comparativas: (1) SAT-Solvers, (2) ataques basados en BDDs, (3) enfoque basado en teoría de grafos y (4) ataques basados en bases de Gröbner.

Este criptoanálisis no utiliza ninguna técnica para agregar restricciones ni agregar algunas ecuaciones, simplemente toma las ecuaciones generadas por Bivium y las transforma a su FNC para después utilizar SAT-Solvers.

Una de las desventajas que tiene este criptoanálisis es que necesitamos verificar la solución que nos entrega el SAT-Solver, ya que puede ser una solución que satisfaga al sistema lineal pero no satisfaga al sistema original.

1.5.2. Criptoanálisis a Bivium con SAT-Solvers de McDonald et al.

El criptoanálisis realizado por McDonald et al. [McDonald et al., 2007] se fundamenta en dos pasos:

1. Producir el sistema de ecuaciones que describe al estado interno de Bivium.
2. Convertir el sistema de ecuaciones obtenidas en un sistema de cláusulas en forma normal conjuntiva.

En el paso 2 los autores, proponen algunas estrategias para realizar el criptoanálisis más rápidamente. Una de ellas consiste en adivinar 34 variables, la razón del porqué intentar adivinar 34 variables es que con ese número de variables adivinadas el tiempo de

ejecución esperado ha resultado el mejor; los tiempos comparativos se obtuvieron de realizar 1,000 experimentos conociendo ciertas variables. Para conocer cuáles 34 variables adivinar, [McDonald et al., 2007] proponen cuatro diferentes estrategias:

Seleccionar las variables que ocurren con mayor frecuencia en el sistema de ecuaciones.

- Seleccionar algunas de las variables que ocurren en los monomios cuadráticos.

Seleccionar todas (o la mayoría) de las variables monomios cuadráticos.

Seleccionar algunas variables alternadas en los productos.

Una desventaja de este criptoanálisis es que es bastante tardado; su tiempo esperado de ejecución es de $2^{42.7}$ segundos. Además, al igual que el trabajo anterior, no agrega ninguna restricción, es por eso que cada respuesta que obtenemos del criptoanálisis debemos verificarla, ya que puede ser una solución a la FNC, pero no una solución para el sistema de ecuaciones original.

1.6. Descripción general de la tesis

Este trabajo está estructurado en cinco capítulos. En este primer capítulo, hemos dado una breve introducción al mundo de la criptografía y con ello, al mundo del criptoanálisis, con el fin de que los objetivos de la tesis sean fácilmente observables. Para poder realizar esto nos apoyamos en los trabajos más recientes en el campo del criptoanálisis. En el capítulo dos mostramos todas las herramientas teóricas necesarias para el entendimiento de la propuesta. En el capítulo tres describimos nuestra propuesta, el criptoanálisis lógico-algebraico; en este capítulo describimos una técnica nueva que hemos desarrollado, llamada *RECNF*; al

final del capítulo mostramos la complejidad del criptoanálisis. En el capítulo cuatro mostramos dos instancias del criptoanálisis lógico-algebraico aplicado a dos diferentes criptosistemas basados en autómatas. En el último capítulo, capítulo cinco, terminamos el trabajo discutiendo los objetivos alcanzados y mostrando tópicos que quedan abiertos para trabajo futuro, con el objetivo de mejorar el criptoanálisis en cuanto a eficiencia.

Capítulo 2

Fundamentos teóricos

En este capítulo mostramos las bases necesarias para la comprensión del criptoanálisis propuesto. Primero, damos una breve introducción a la criptografía y el criptoanálisis. En la sección 2 mostramos un poco de matemáticas básicas para criptografía. En la sección 4, damos una breve introducción a los registros de desplazamiento con retroalimentación lineal (conocidos como LFSR). En la sección 5, una pequeña introducción a los autómatas finitos; en esta sección también exponemos los conceptos utilizados por [Tao, 2008] sobre autómatas débilmente invertibles y sus inversos. Posteriormente, en la sección 6, mostramos las funciones booleanas y su utilidad dentro de la criptografía. En la sección 8, presentamos algunos de los algoritmos que han sido propuestos para resolver sistemas de ecuaciones polinómicas. En la sección 9 describimos el algoritmo para convertir un sistema de ecuaciones polinómicas multivariable a la forma normal conjuntiva.

2.1. Criptografía y terminología

El término más general en el área es la criptología y no la criptografía, como se pudiera llegar a pensar. En la Figura 2.1 se muestra que la criptología se divide en dos ramas

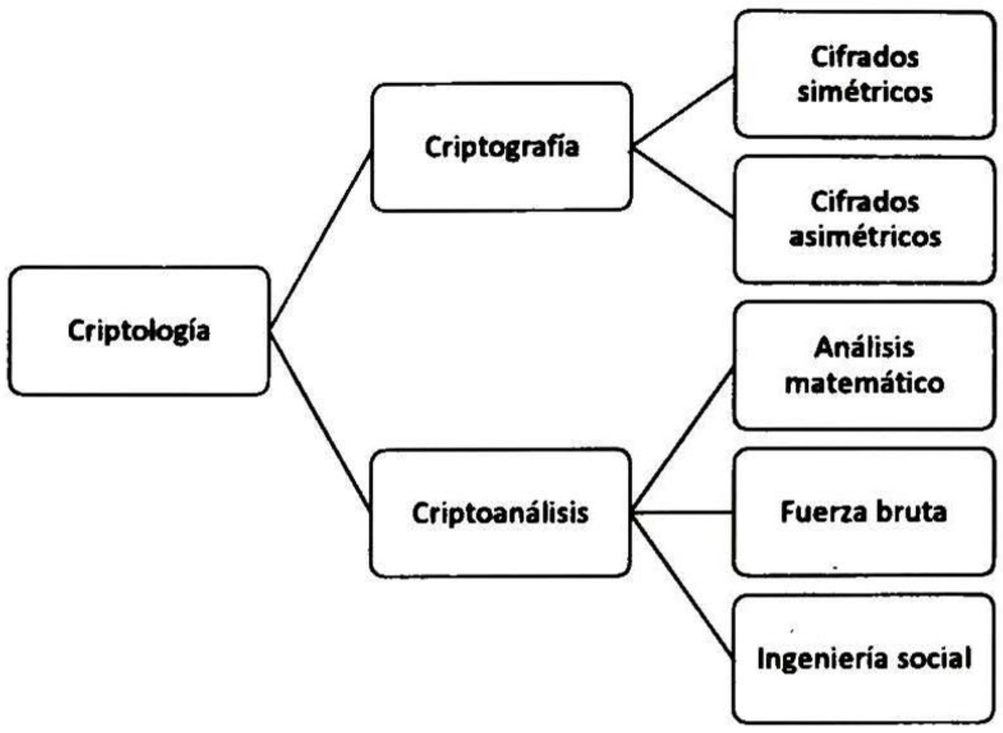


Figura 2.1: Ramificación de la criptología

principales:

- **Criptografía**: Es la ciencia de la escritura secreta con la meta principal de esconder el significado del mensaje.

- **Criptoanálisis**: Es la ciencia de *romper* el criptosistema (descifrar el mensaje sin conocer la clave privada o encontrar la clave privada). El criptoanálisis es de mucha importancia para los criptosistemas modernos, ya que sin gente que intente romper el criptosistema, nunca nos podríamos dar cuenta si realmente es seguro o no.

Dentro del criptoanálisis nosotros nos enfocaremos en el área clásica y en el análisis matemático.

2.1.1. Criptosistemas

Principalmente existen dos enfoques para la criptografía: (1) Criptografía de clave privada, (2) Criptografía de clave pública. A continuación se da una breve explicación de cada enfoque.

Criptografía de clave privada (simétrica)

La criptografía de clave privada cuenta con un algoritmo de encriptación y de desencriptación; estos dos algoritmos necesitan la misma clave privada para funcionar correctamente. En la Figura 2.2 se muestra, a grandes rasgos, la criptografía de clave privada.

La criptografía de clave privada cuenta con 5 elementos:

1. **Texto plano**: Es el mensaje original. Es la entrada para el algoritmo de encriptación.

2. **Algoritmo de encriptación**: Este algoritmo realiza algunas transformaciones sobre el texto plano.

3. **Clave privada**: También es la entrada al algoritmo de encriptación. Las transformaciones que realice el algoritmo de encriptación al texto plano dependen de la clave privada.

4. **Texto cifrado**: Es la salida producida por el algoritmo de encriptación.

5. **Algoritmo de desencriptación**: Esencialmente es el algoritmo de encriptación funcionando al revés. Este algoritmo toma como entrada el texto cifrado y la clave privada, la salida del algoritmo de desencriptación es el texto plano.

Sea α el mensaje (texto plano), κ la clave privada, E una función de encriptado y D una función de desencriptado; la *criptografía de clave privada* se puede expresar de la siguiente forma:

$$\alpha = D(E(\alpha, \kappa), \kappa) \tag{2.1}$$

Es importante mencionar que la seguridad de la criptografía de clave privada depende de la secrecía de la clave privada y no del algoritmo de encriptación.

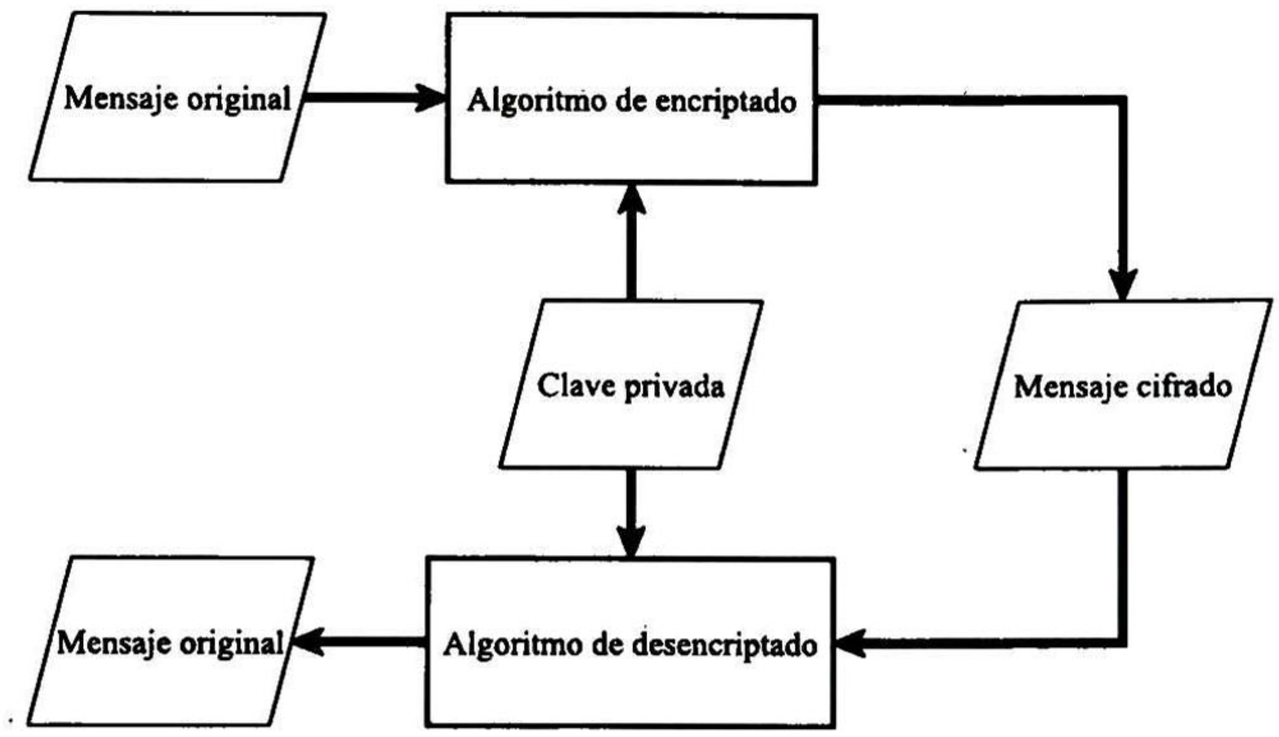


Figura 2.2: Criptografía de clave privada

Criptografía de clave pública (asimétrica)

La criptografía de clave pública necesita un algoritmo de encriptación y un algoritmo de desencriptación, la principal diferencia con la clave privada, es que la criptografía con clave pública necesita de dos claves, una privada y una pública (de ahí es de donde viene el término). La Figura 2.3 muestra, a grandes rasgos, la criptografía de clave pública.

La criptografía de clave pública cuenta con 6 elementos:

1. **Texto plano**: Es el mensaje original. También es la entrada al algoritmo de encriptación.

2. **Texto cifrado**: Es la salida del algoritmo de encriptación.

3. **Algoritmo de encriptación**: Realiza diferentes transformaciones al texto plano, estas transformaciones dependen tanto del texto plano como de la clave pública; la salida de este algoritmo es el texto cifrado.

4. **Algoritmo de desencriptación**: Realizada diferentes transformaciones al texto cifrado, tiene como entrada el texto cifrado y la clave privada; la salida del algoritmo de

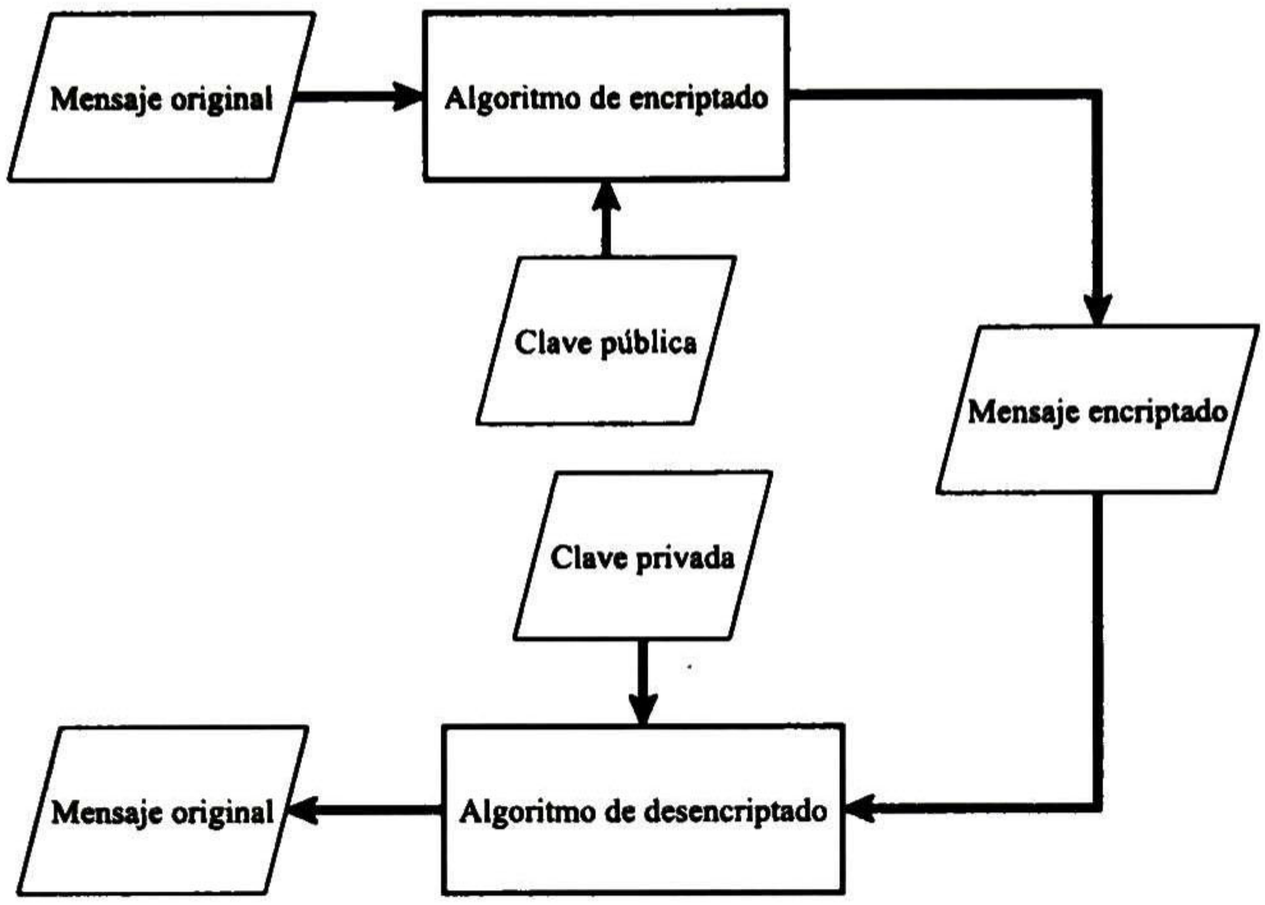


Figura 2.3: Criptografía de clave pública

desencriptación es el texto plano.

5. **Clave privada**: Es la clave que sólo es conocida por el usuario receptor.

6. **Clave pública**: Es la clave que el receptor hace pública para que los usuarios que desean comunicarse con él puedan encriptar los mensajes dirigidos hacia él.

La criptografía de clave pública, a diferencia de la criptografía con clave privada, cuenta con dos claves (la clave pública y la clave privada), estas dos claves tienen cierta relación una con la otra. A diferencia de la criptografía con clave privada, la criptografía de clave pública necesita de las dos claves (y no de una sola) para poder descifrar el mensaje.

Sea α el mensaje original (texto plano), κ_{pu} la clave pública, κ_{pr} la clave privada, E el algoritmo de encriptación y D el algoritmo de desencriptación; entonces la criptografía de clave pública se puede expresar de la siguiente manera:

$$\alpha = D(E(\alpha, \kappa_{pu}), \kappa_{pr}) \tag{2.2}$$

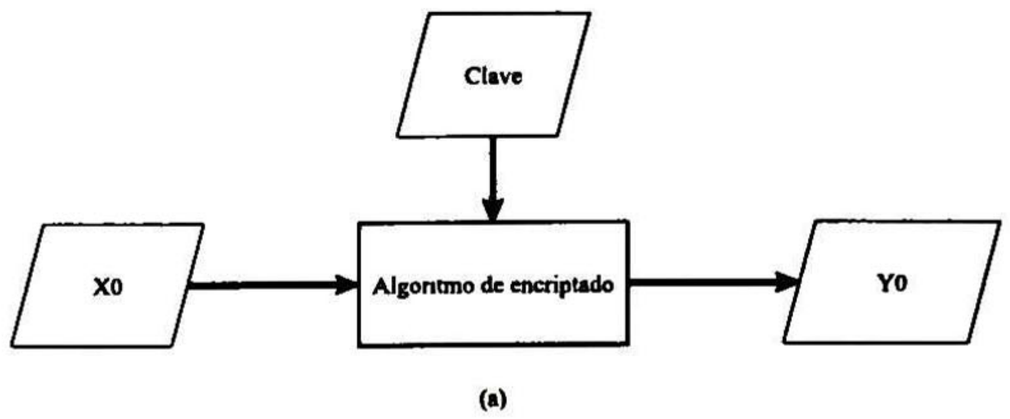


Figura 2.4: Tipos de cifrado. (a) Cifrado de flujo. (b) Cifrado de bloques

Tipos de cifrado

Los cifrados se pueden hacer de dos maneras: cifrados de flujo o cifrados por bloque. Los cifrados de flujo encriptan los bits individualmente; la Figura 2.4 (a) muestra un cifrado de flujo, el cual va cifrando bit por bit. Los cifrados por bloque encriptan **b** bits, con $b \in \mathbb{N}$ y $b > 0$, y la salida son c bits (no necesariamente $c = b$) con $c \in \mathbb{N}$ y $c > 0$; la Figura 2.4 (b) muestra un cifrado por bloques.

2.1.2. Tipos de ataques a los cifrados

Existen diferentes tipos de ataques a los cifrados de flujo, se conocen como: (1) ataque de texto cifrado, (2) ataque de texto plano conocido, y (3) ataque de texto seleccionado. Según los datos que el criptoanalista puede utilizar para romper el criptosistema es el tipo de ataque que se debe realizar. Los datos que se pueden conocer según un ataque conocido

son los siguientes:

1. **Ataque de texto cifrado**. Los datos que el criptoanalista conoce son:

   - el algoritmo de encriptación,
   - texto cifrado.

2. **Ataque de texto plano conocido**. Los datos que el criptoanalista conoce son:

   - el algoritmo de encriptación,
   - texto cifrado,
   - uno o mas pares de texto plano-texto cifrado, formados con la clave secreta.

3. **Ataque de texto seleccionado**. Los datos que el criptoanalista conoce son:

   - el algoritmo de encriptación,
   - texto cifrado,
   - texto plano seleccionado junto con el texto cifrado correspondiente (generado con la clave secreta).

En este trabajo consideraremos el tipo de ataque de ***texto seleccionado***.

2.2. Criptoanálisis

El criptoanálisis es la ciencia de recuperar la clave privada a partir del texto cifrado, o el texto plano del texto cifrado. El criptoanálisis tiene tres ramificaciones principales (como se muestra en la Figura 2.5): (1) matemático, (2) fuerza bruta, (3) ingeniería social. El

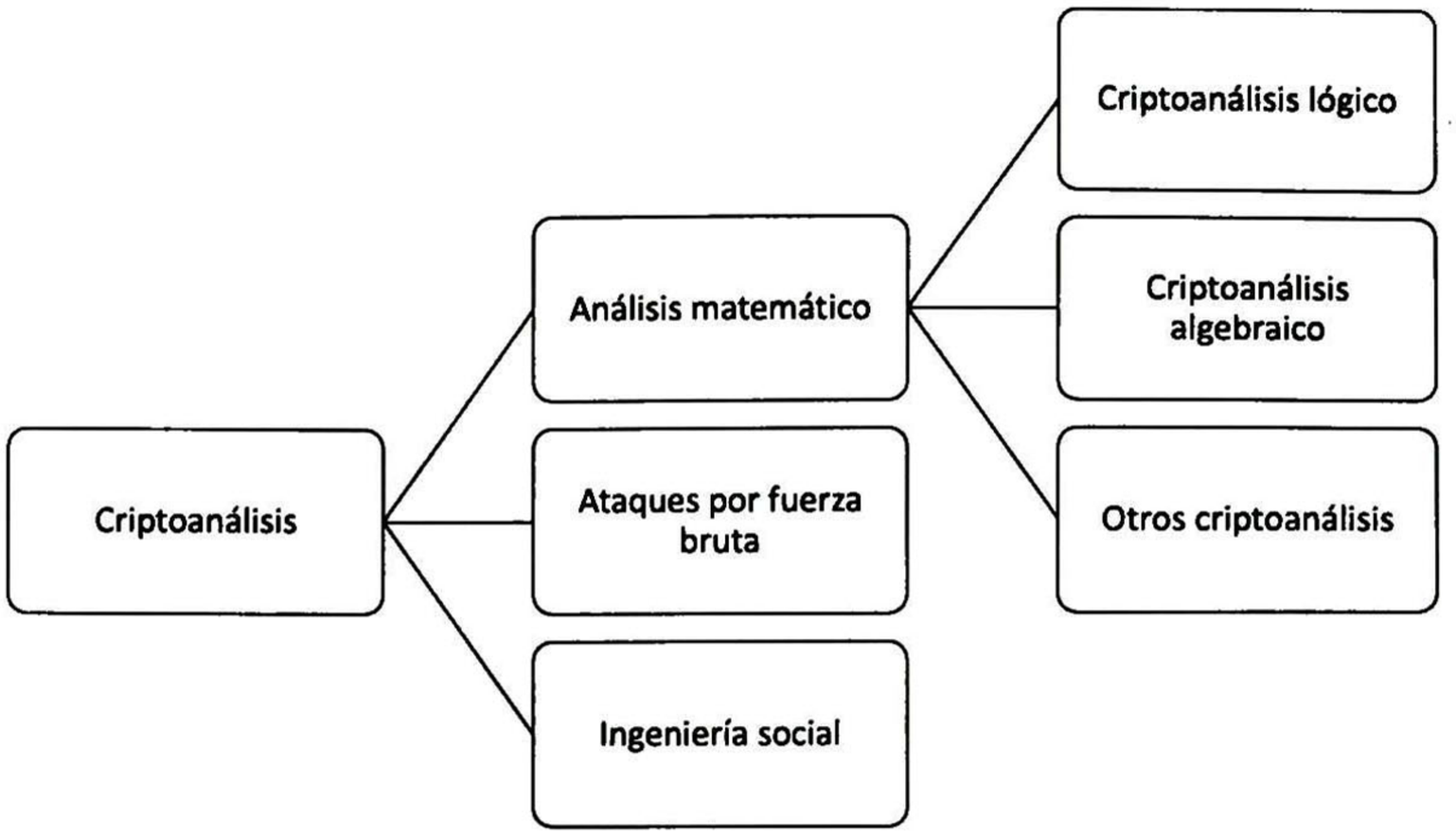


Figura 2.5: Ramificación del criptoanálisis

criptoanálisis en que nos enfocamos en esta tesis es el criptoanálisis matemático, esta manera de hacer criptoanálisis requiere el conocimiento previo de la estructura del método de cifrado.

También tenemos los otros dos tipos de criptoanálisis, los cuales solamente describiremos brevemente: (1) criptoanálisis por fuerza bruta, consiste en probar todas las posibles claves de método de cifrado hasta obtener la clave privada; (2) criptoanálisis por ingeniería social, consiste en obtener la clave privada por medio de la persona que conoce la clave privada, es decir, se vale de utilizar maltrato, amenazas, soborno, espionaje, etc., hacia la persona que conoce la clave privada hasta nos la proporcione.

Actualmente el criptoanálisis matemático tiene muchas variantes, en esta tesis sólo nos enfocaremos al criptoanálisis algebraico y al criptoanálisis lógico.

2.2.1. Criptoanálisis lógico

Este criptoanálisis fue propuesto inicialmente por Massacci y Marraro [Massacci y Marraro, 2000]. En su propuesta presentan un caso de estudio donde criptoanali-

zan a DES: primero codifican sus propiedades en fórmulas lógicas, después buscan un modelo que satisfaga a las fórmulas y decodifican las propiedades según el modelo obtenido.

En general, dado un modelo de cifrado particular, el criptoanálisis lógico consiste en primero codificar las propiedades de los algoritmos de cifrado como problemas de satsifacibilidad y luego utilizar sistemas de prueba automática de teoremas y SAT-Solvers para poder obtener la clave privada o el texto plano, según sea el caso.

2.2.2. Criptoanálisis algebraico

El criptoanálisis algebraico fue propuesto primero por Courtois y Meier [Courtois y Meier, 2003], dentro de su propuesta analizan como modelar al sistema de encriptación TOYOCRYPT y al sistema de encriptación LILI-128 (ambos son cifrados de flujo).

En general, el criptoanálisis algebraico consiste en modelar algebraicamente el criptosistema en un sistema de ecuaciones polinomial y después resolver este sistema de ecuaciones; una vez resuelto este sistema de ecuaciones obtenemos la clave privada o el texto cifrado, dependiendo de como modelamos el criptosistema.

Analizar, por medio del criptoanálisis algebraico, a un sistema de cifrado consiste básicamente de dos pasos:

(1) Convertir el cifrado y posiblemente alguna información adicional, en un sistema de ecuaciones polinómicas, normalmente sobre el campo $GF(2)$ (vea la Definición 2.3.4).

(2) Resolver el sistema de ecuaciones polinómicas multivariable que obtuvimos para recuperar la clave privada o el texto plano, según sea el caso que modelamos.

2.3. Matemáticas para criptografía

En esta sección mostramos algunos de los conceptos aritméticos y algebraicos que se emplean en esta tesis. Se explicará de una manera sucinta la aritmética modular, las definiciones de grupo, anillo y campo, por último damos una introducción a los campos de Galois. Para mayor información se recomienda [Smart, 2003].

2.3.1. Aritmética modular

Sea N un número entero positivo, el cuál llamamos *módulo*. Sean a y b dos enteros cualesquiera. Escribimos $a \equiv b \pmod{N}$ si N divide $b-a$, y decimos que a y b son *congruentes* módulo N.

También consideramos el operador sufijo $(mod\ N)$ en un entero como la operación que devuelve el valor no negativo mas pequeño igual al módulo N.

La relación binaria es *congruente a módulo* N, es de equivalencia e induce una partición del conjunto $\mathbb{Z}$ de los números enteros en N clases de equivalencia (llamadas *clases residuales*), exactamente. Si representamos cada una de estas clases por el menor entero no negativo que contiene, entonces tenemos el conjunto cociente de $\mathbb{Z}$ bajo esta relación:

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}. \tag{2.3}$$

2.3.2. Grupos, anillos y campos finitos

Definición 2.3.1 (Grupo). Un *grupo* es un conjunto no vacío $(\mathbb{G})$ con una operación binaria $(\bullet)$ que cumple las siguientes propiedades:

- Es cerrada:

$$\text{Para todo } a, b \in \mathbb{G} : a \bullet b \in \mathbb{G}.$$

- Es asociativa:

$$\text{Para todo } a, b, c \in \mathbb{G} : (a \bullet b) \bullet c = a \bullet (b \bullet c).$$

- Admite un elemento identidad (denotado como 1):

$$\text{Para todo } a \in \mathbb{G} : a \bullet 1 = 1 \bullet a = a.$$

- Cada elemento tiene inverso:

$$\text{Para todo } a \in \mathbb{G}, \text{existe } a^{-1} \in \mathbb{G} : a \bullet a^{-1} = 1.$$

Denotamos a un grupo como la dupla $(\mathbb{G}, \bullet)$.

Si además la operación $\bullet$ del grupo $\mathbb{G}$ conmutativa, esto es:

$$\text{Para todo } a, b \in \mathbb{G} : a \bullet b = b \bullet a.$$

entonces decimos que G es un *grupo abeliano* (o *conmutativo*).

Definición 2.3.2 (Anillo con identidad)**.** Un *anillo con identidad* es una tripleta $\mathcal{K} = (\mathbb{K}, +, \cdot)$, donde $\mathbb{K}$ es un conjunto no vacío y $+$, $\cdot$ son operaciones binarias en $\mathbb{K}$ tales que:

$(\mathbb{K}, +)$ es un grupo abeliano con identidad denotada por 0;

$\cdot$ es una operación binaria cerrada, asociativa y hay un elemento identidad en $\mathbb{K}$ bajo que denotamos por 1;

- $\cdot$ es distributiva con respecto a $+$:

  Para todos los $a, b, c \in \mathbb{K} : a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ y $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$.

Si además la operación es conmutativa, decimos que el *anillo $\mathcal{K}$ es conmutativo con identidad.*

Nota: cuando no hay ambigüedad, el símbolo se omite y la operación se muestra en yuxtaposición.

Definición 2.3.3 (Campo). Sea $\mathcal{K} = (\mathbb{K}, +, \cdot)$ un anillo conmutativo con identidad. Decimos que $\mathcal{K}$ es un *campo* si todos los elementos distintos del elemento identidad bajo $+$ (0) tienen inverso, esto es:

- Para cada $a \in \mathbb{K} - \{0\}$, existe $b \in \mathcal{K}$ tal que $a \cdot b = 1$; como b resulta ser único para a se denota como a^{-1}

Definición 2.3.4 (Campos de Galois). Si $\mathcal{K} = (\mathbb{K}, +, \cdot)$ es un campo finito con p^n elementos, para algún primo p y algún entero positivo n, entonces es llamado un *campo de Galois* y denotado como $GF(p^n)$.

Los campos de Galois también son denotados como $\mathbb{F}_{q^m}$, donde q es un número primo y $m \in \mathbb{N}$.

Denotamos como $\mathbb{F}_m^n$ el conjunto de n elementos de $\mathbb{F}_m$, esto es:

$$\mathbb{F}_m^n = \{a_1, \ldots, a_n\}$$

donde $a_i \in \mathbb{F}_m$, para $i = 1, \ldots, n$

En esta tesis sólo utilizaremos $GF(2) = \mathbb{Z}_2$.

2.3.3. Polinomios sobre anillos

Sea $\mathbb{R}$ un anillo conmutativo con identidad. Un polinomio $p(x)$ en la indeterminada sobre el anillo $\mathbb{R}$ es una expresión de la forma

$$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

donde $a_n \in \mathbb{R}$ y $n \in \mathbb{N}$.

El elemento a_i es llamado el coeficiente de x^i en $p(x)$. Si el coeficiente de a_ix^i es 0, el término a_ix_i suele omitirse, y si el coeficiente es 1, suele expresarse como x^i simplemente.

Definición 2.3.5 (Polinomios univariable sobre anillos)**.** Sea $\mathcal{K}$ un anillo conmutativo con identidad. El conjunto de todos los polinomios en x con coeficientes del anillo $\mathcal{K}$ se denota por $\mathcal{K}[x]$. Esto es:

$$\mathcal{K}[x] := \{a_nx^n + a_{n-1}x^{n-1} + \cdot + a_1x + a_0 \mid a_i \in \mathcal{K}, n \in \mathbb{N}\}.$$

Definición 2.3.6 (Polinomios multivariable sobre anillos)**.** Sea $\mathcal{K}$ un anillo conmutativo con identidad y sea $\{x_1, x_2, \ldots, x_n\}$ un conjunto de variables. Entonces denotamos como $\mathcal{K}[x_1, \ldots, x_n]$ el conjunto de todos los polinomios en $x_1, x_2, \cdots, x_n$ con coeficientes en $\mathcal{K}$. Esto es, el conjunto de todas las expresiones formales de la forma:

$$\sum_{\alpha=(\alpha_1,\ldots,\alpha_n)} a_\alpha x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

donde $a_\alpha \in \mathcal{K}$, la sumatoria corre en

$$\alpha = (\alpha_1, \ldots, \alpha_n) \in (\mathbb{N} \cup \{0\})^n$$

y, a lo más, un número finito de a_α son diferentes de cero.

Es conveniente utilizar una notación mas simple para un polinomio multivariable cualquiera, como $f(x_1, \ldots, x_n), g(x_1, \ldots, x_n)$ y así. Es decir

$$f(x_1, \ldots, x_n) = \sum_{\alpha} a_\alpha x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

para alguna $a_\alpha \in \mathcal{K}$.

Ejemplo 2.3.1. Sea $\mathcal{K} = (\mathbb{Z}_7, +, \cdot)$ y $\mathbb{Z}_7[x_1, x_2, x_3, x_4, x_5, x_6]$. Entonces

$$6x_1x_2 + 3x_3x_6 + 5x_4 + 2x_5$$

$$3x_1 + 2x_2x_6 + 6x_3. \quad \square$$

2.4. Registros de desplazamiento con retroalimentación lineal (LFSR)

Los registros de desplazamiento con retroalimentación lineal (LFSR, por su acrónimo en inglés *Linear Feedback Shift Register*) consisten de unidades de almacenamiento (normalmente flip-flops), también conocidas como estados.

De ahora en adelante para representar el *i-ésimo* estado de un LFSR lo haremos por medio de la notación s_i. El número de unidades de almacenamiento nos da el grado del LFSR.

Matemáticamente podemos modelar un LFSR con m estados (también conocido como *LFSR de grado m*) con los coeficientes de retroalimentación $p_0, p_1, \ldots, p_{m-1}$ de la siguiente manera:

$p_i = 0$, si la salida del estado s_i no es utilizada para la retroalimentación,

$i = 0, 1, \ldots, m-1$.

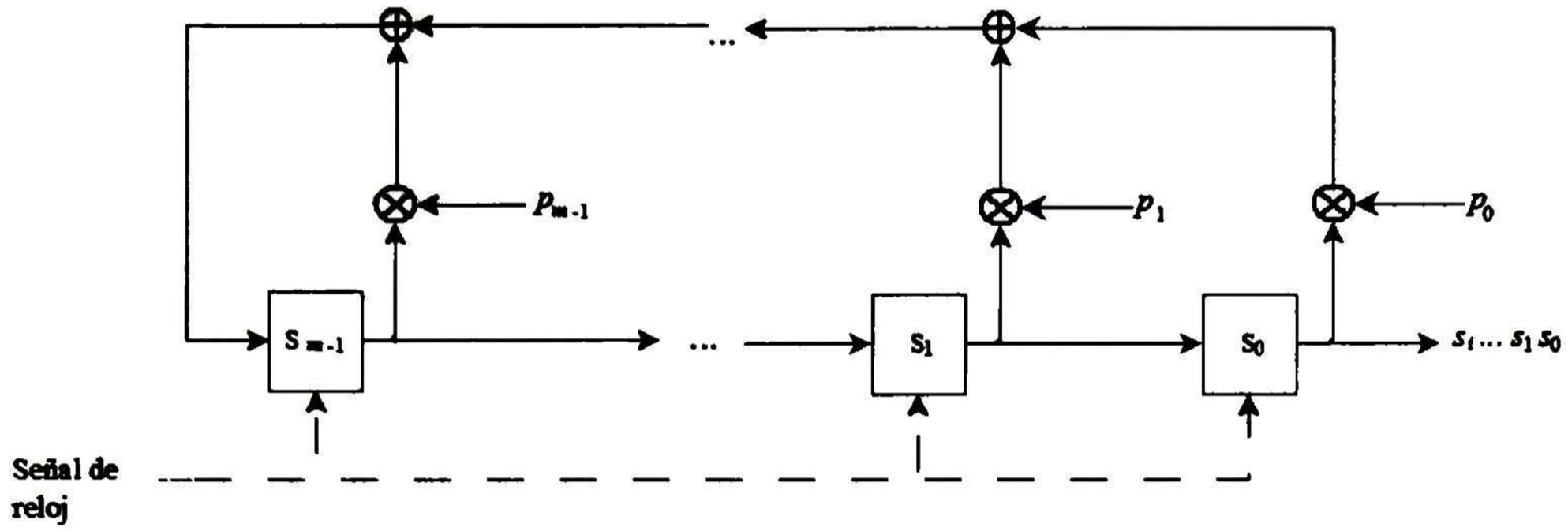


Figura 2.6: Un LFSR de grado m con sus coeficientes de retroalimentación p_i

- $p_i = 1$, si la salida del estado es utilizada para la retroalimentación, $i = 0, 1, \ldots, m-1$.

La Figura 2.6 muestra en general un LFSR de grado m y sus coeficientes de retroalimentación.

La secuencia de salida de un LFSR de grado m puede ser calculada con la siguiente fórmula matemática:

$$s_{i+m} \equiv \sum_{j=0}^{m-1} (p_j \cdot s_{i+j}) \ (mod\ 2),\ i = 0, 1, 2, \ldots \tag{2.4}$$

donde $s_i, p_j \in \{0, 1\}$.

Es decir, podemos calcular el estado j-ésimo del LFSR, para todo $j > m$, conociendo los estados s_i, para todo $i = 0, 1, ..., j-1$, y conociendo todas las $p_i, i = 0, 1, \ldots, m-1$.

Ya que un LFSR de grado m sólo tiene un número finito de estados (m estados para ser exactos), la secuencia de salida de un LFSR se repite periódicamente después de cierto estado; a toda la secuencia de salidas generada sin que se repita se le conoce como *periodo*; en otras palabras, a toda secuencia de salida $s_0, s_1, \ldots, s_n$ le llamaremos periodo si $s_{n+1} = s_0, s_{n+2} = s_1, \ldots, s_{2n} = s_n$. El *periodo máximo* que puede tener un LFSR de grado m es $2^m - 1$ [Paar y Pelzl, 2009].

2.5. Autómatas finitos

En esta sección damos una introducción a los autómatas finitos, también conocidos como máquinas de Mealy o máquinas transductoras simples, a las que de ahora en adelante simplemente llamaremos autómatas.

Un autómata es una séxtupla $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$, donde Σ es un conjunto finito de *símbolos de entrada*, Γ es un conjunto finito de *símbolos de salida*, S es un conjunto finito de *estados*, $s_0 \in S$ es el *estado inicial*, $\delta : S \times \Sigma \to S$, es la *función de transición* y $\lambda : S \times \Sigma \to \Gamma$ es la *función de salida*.

Para cualquier conjunto no vacío A, denotamos como A^* al conjunto de todas las palabras (secuencias finitas) sobre A. Denotamos como ε a la palabra vacía. Extendemos el dominio de la función δ a $S \times \Sigma^*$ de la siguiente manera:

$$\begin{aligned} \delta : S \times \Sigma^* &\to S, \\ \bar{\delta}(s, \varepsilon) &:= s, \\ \bar{\delta}(s, x\alpha) &:= \delta(\bar{\delta}(s, \alpha), x), \end{aligned}$$

para cada $s \in S$, $\alpha \in \Sigma^*$ y $x \in \Sigma$.

De igual manera extendemos el dominio de la función $\bar{\lambda}$ a $S \times \Sigma^*$:

$$\begin{aligned} \bar{\lambda} : S \times \Sigma^* &\to \Gamma, \\ \bar{\lambda}(s, \varepsilon) &:= \varepsilon, \\ \bar{\lambda}(s, x\alpha) &:= \lambda(s, x)\bar{\lambda}(\delta(s, x), \alpha). \end{aligned}$$

para cada $s \in S$, $\alpha \in \Sigma^*$ y $x \in \Sigma$.

2.5.1. Autómata con memoria de orden-(h, k)

Sean h y k enteros no negativos, Σ y Γ conjuntos no vacíos de símbolos, $f : \Sigma^{h+1} \times \Gamma^k \to \Gamma$ una función, $x_{-h}, x_{-h+1}, \ldots, x_{-1}, x_0 \in \Sigma$, $y_{-k}, y_{-k+1}, \ldots, y_0 \in \Gamma$ y $y_0 = f(y_{-1}, y_{-2}, \ldots, y_{-k}, x_0, x_{-1}, \ldots, x_{-h})$. Definimos al *autómata con memoria de orden-(h,k)*, determinado por f como:

$$\begin{aligned} M_f &= \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle, \\ s_0 &= \langle y_{-1}, \ldots, y_{-k}, x_{-1}, \ldots, x_{-h} \rangle \\ \delta(\langle y_{i-1}, \ldots, y_{i-k}, x_{i-1}, \ldots, x_{i-h} \rangle, x_i) &= \langle y_i, \ldots, y_{i-k+1}, x_i, \ldots, x_{i-h+1} \rangle, \\ \lambda(\langle y_{i-1}, \ldots, y_{i-k}, x_{i-1}, \ldots, x_{i-h} \rangle, x_i) &= y_i \\ y_i &= f(y_{i-1}, y_{i-2}, \ldots, y_{i-k}, x_i, x_{i-1}, \ldots, x_{i-h}) \end{aligned}$$

y se denota como $M_f = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$, donde $S = \Gamma^k \times \Sigma^h$ y $s_0 \in S$. A M_f se le conoce como el *autómata inducido por f.*

Casos especiales del autómata finito con memoria de orden-*(h,k)*

Si $k = 0$, entonces al autómata M_f se le conoce como *autómata finito con memoria de entrada de orden h.* Si $h = 0$, entonces, al autómata se le conoce como *autómata finito con memoria de salida k.*

Ejemplo 2.5.1. Sea $\Sigma = \Gamma = \{0, 1\}$. Sean $x_i \in \Sigma$, para $i = -3, -2, -1, 0...$ y $y_i \in \Gamma$, para $i = -1, 0, ...$ Sea $f : \Gamma^1 \times \Sigma^4 \to \Gamma$ una función definida de la siguiente manera:

$$f(x_0, x_1, x_2, x_3, y_1) = x_0 + x_1 x_2 + x_3 y_1. \tag{2.5}$$

El autómata con memoria de orden-(3,1) inducido por f queda definido de la si-

guiente manera:

$$\begin{aligned} M_f &= \langle \Sigma, \Gamma, \Gamma^1 \times \Sigma^3, s_0, \delta, \lambda \rangle, \\ s_0 &= \langle y_{-1}, x_{-1}, x_{-2}, x_{-3} \rangle \\ \delta(\langle y_{i-1}, x_{i-1}, x_{-2}, x_{i-3} \rangle, x_i) &= \langle y_i, x_i, x_{i-1}, x_{i-2} \rangle, \\ \lambda(\langle y_i, x_{i-1}, x_{i-2}, x_{i-3} \rangle, x_i) &= y_i, \\ y_i &= f(x_i, x_{i-1}, x_{i-2}, x_{i-3}, y_{i-1}) \end{aligned}$$

para, $i = 0, 1, ...$ □

2.5.2. Composición de autómatas

La composición de autómatas se puede hacer de dos maneras: (1) en serie, (2) en paralelo.

Composición en serie

Sean $M_1 = \langle \Sigma, \Gamma, S_1, s_0, \delta_1, \lambda_1 \rangle$ y $M_2 = \langle \Gamma, \Upsilon, S_2, s'_0, \delta_2, \lambda_2 \rangle$ dos autómatas. Definimos al *autómata compuesto en serie* como:

$$C(M_1, M_2) = \langle \Sigma, \Upsilon, S_1 \times S_2, \langle s_0, s'_0 \rangle, \delta, \lambda \rangle$$

donde

$$\begin{aligned} \delta(\langle s_1, s_2 \rangle, x) &= \langle \delta_1(s_1, x), \delta_2(s_2, \lambda_1(s_1, x)) \rangle, \\ \lambda(\langle s_1, s_2 \rangle, x) &= \lambda_2(s_2, \lambda_1(s_1, x)), \end{aligned}$$

para todos $s_1, s_0 \in S_1, s_2, s'_0 \in S_2, x \in \Sigma$.

Esta composición es el resultado de introducir la salida del primer autómata M_1 en la entrada del segundo autómata M_2.

Composición en paralelo

Sean $M_1 = \langle \Sigma_1, \Gamma_1, S_1, s_0, \delta_1, \lambda_1 \rangle$ y $M_2 = \langle \Sigma_2, \Gamma_2, S_2, s_0', \delta_2, \lambda_2 \rangle$ dos autómatas. Definimos al autómata compuesto en paralelo como:

$$P(M_1, M_2) = \langle \Sigma_1 \times \Sigma_2, \Gamma_1 \times \Gamma_2, S_1 \times S_2, \langle s_0, s_0' \rangle, \delta, \lambda \rangle$$

donde

$$\begin{aligned} \delta(\langle s_1, s_2 \rangle, (x_1, x_2)) &= \langle \delta_1(s_1, x_1), \delta_2(s_2, x_2) \rangle, \\ \lambda(\langle s_1, s_2 \rangle, (x_1, x_2)) &= \langle \lambda_1(s_1, x_1), \lambda_2(s_2, x_2) \rangle, \end{aligned}$$

para todos los $s_1, s_0 \in S_1, s_2, s_0' \in S_2, x_1 \in \Sigma_1, x_2 \in \Sigma_2$.

El objetivo de esta composición es hacer que un sólo autómata calcule las salidas de M_1 y M_2.

2.5.3. Autómata débilmente invertible

Sea $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$ un autómata. Se dice que M *es débilmente invertible* si y sólo si para cualesquiera $s \in S$ y $\alpha, \alpha' \in \Sigma^*$, si $\lambda(s, \alpha) = \lambda(s, \alpha')$ entonces $\alpha = \alpha'$. Es decir, M es débilmente invertible si y sólo si para cualquier $s \in S$ y cualquier $\alpha \in \Sigma^*$ α puede ser determinada unívocamente por s y $\lambda(s, \alpha)$. En [Tao, 2008] se encuentra la siguiente caracterización de los autómatas débilmente invertibles.

Lema 2.5.1 (Autómata débilmente invertible [Tao, 2008]). Sea $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$ un autómata. M es débilmente invertible si y sólo si para cualesquiera $s_0 \in S$, $x, x' \in \Sigma$ y $\alpha, \alpha' \in \Sigma^*$, si $\lambda(s_0, x\alpha) = \lambda(s_0, x', \alpha')$, entonces $x = x'$.

Sea τ un entero no negativo. Un autómata $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$ se dice que es *invertible con retraso* τ, si para cualquier $s_0 \in S$ y cualesquiera $x_i \in \Sigma, i = 0, 1, \ldots, \tau, x_0$

puede ser determinado unívocamente por $\lambda(s_0, x_0 \dots x_\tau)$; es decir, si para cualesquiera $s_1, s_2 \in S$ y cualesquiera $x_i, x_i' \in \Sigma, i = 0, 1, \dots, \tau$, si $\lambda(s_1, x_0 \dots x_\tau) = \lambda(s_2, x_0' \dots x_\tau')$ entonces $x_0 = x_0'$.

2.5.4. Autómata inverso débil

Sean $M = \langle \Sigma, \Gamma, S_1, s_0, \delta_1, \lambda_1 \rangle$ y $M' = \langle \Gamma, \Sigma, S_2, s_0', \delta_2, \lambda_2 \rangle$ dos autómatas. Sean $\alpha \in \Sigma^*, \alpha_0 \in \Sigma^*, s_0 \in S_1, s_0' \in S_2$. Si para cualquier α existe un α_0 con longitud τ tal que $\lambda_2(s_0', \lambda_1(s_0, \alpha)) = \alpha_0\alpha$, entonces al par (s_0, s_0') se le llama *par compatible con retraso* τ, también se dice que s_0' *es compatible con* s_0 *con retraso* τ. En otras palabras, si componemos M y M' en serie $(C(M, M'))$ y dejamos pasar τ salidas de la máquina compuesta obtenemos la misma entrada.

Si para cualquier $s_0 \in S_1$ existe un $s_0' \in S_2$ tal que (s_0, s_0') es un par compatible con retraso τ, entonces decimos que M' *es la inversa débil de* M.

2.5.5. Autómatas lineales

En esta subsección damos una introducción sucinta de lo presentado por Kohavi en [Kohavi y Jha, 2010], capítulo 15.

Sean k, l y m enteros no negativos. Sea q un número primo. Sea $Y_i \in \mathbb{F}_2^k$, para $i = 0, 1, \dots$. Consideremos un autómata lineal de dimensión k sobre $GF(q)$, con l entradas y m salidas. El estado siguiente Y_i puede ser expresado como una función de las entradas de la máquina y el estado actual como sigue:

$$Y_i = \sum_{j=1}^{k} (\alpha_{ij} y_j) + \sum_{j=1}^{l} (\beta_{ij} x_j)$$

donde x_i es la i-ésima entrada, para $i = 1, 2, \ldots, l$, y_j es el estado j-ésimo, para $j = 0, 1, \ldots, m$, α_{ij} es distinto de 0 si el estado y_j del autómata se conecta con la salida Y_i, β_{ij} es distinto de 0 si el estado x_j del autómata está conectada a la salida Y_i (ver las páginas 539-540 de [Kohavi y Jha, 2010]).

Todo el conjunto de ecuaciones de estado siguiente se puede expresar de forma matricial como sigue:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \ldots & \alpha_{1k} \\ \alpha_{21} & \alpha_{22} & \ldots & \alpha_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{k1} & \alpha_{k2} & \ldots & \alpha_{kk} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} + \begin{bmatrix} \beta_{11} & \beta_{12} & \ldots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \ldots & \beta_{2l} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{k1} & \beta_{k2} & \ldots & \beta_{kl} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix} \tag{2.6}$$

Podemos simplificar la representación anterior de la siguiente manera:

$$Y(t) = y(t+1) = Ay(t) + Bx(t) \tag{2.7}$$

donde

$$Y(t) = y(t+1) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}, A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \ldots & \alpha_{1k} \\ \alpha_{21} & \alpha_{22} & \ldots & \alpha_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{k1} & \alpha_{k2} & \ldots & \alpha_{kk} \end{bmatrix}, B = \begin{bmatrix} \beta_{11} & \beta_{12} & \ldots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \ldots & \beta_{2l} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{k1} & \beta_{k2} & \ldots & \beta_{kl} \end{bmatrix},$$

$$y(t) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}, x(t) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix}$$

El vector $y(t)$ es llamado el *vector de estado actual en el tiempo t*, el vector $Y(t)$ es llamado el *vector de estado siguiente en el tiempo t*, el vector $x(t)$ es llamado el *vector de entrada en el tiempo t*

Definición 2.5.1 (Autómata lineal). Se dice que un *autómata es lineal* sobre un campo finito $GF(q)$ si sus estados pueden ser identificados con los elementos de un vector y tanto el estado siguiente como la función de salida pueden ser especificados por matrices sobre $GF(q)$.

2.6. Funciones booleanas

Sea n un número entero positivo. Sea $f : \{0,1\}^n \to \{0,1\}$ una función booleana de n variables. La tabla de f se puede ver como una tabla de verdad con 2^n filas y $n+1$ columnas. También podemos representar a f como un polinomio en n variables sobre $\mathbb{F}_2$ (llamadas *variables booleanas*). Este polinomio puede ser expresado como una suma de productos, distintos entre sí. Más precisamente, $f(x_1, \ldots, x_n)$ puede escribirse de la forma:

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_u \left(\prod_{i=1}^{n} x_i^{u_i} \right), a_u \in \mathbb{F}_2, u = (u_1, \ldots, u_n). \tag{2.8}$$

La expresión (2.8) es conocida como la *forma normal algebraica* (*FNA*) de la función f.

El Algoritmo 1 nos muestra un procedimiento para obtener la FNA a partir de una tabla de verdad.

Ejemplo 2.6.1 (Obtención de la FNA a partir de una tabla de verdad). Supongamos que sólo conocemos la función f_1 por su tabla (Tabla 2.1) y queremos obtener su FNA. Entonces, aplicando el Algoritmo 1 obtenemos:

$$f_1(x_1, x_2, x_3) = x_1 \oplus x_2 x_3 \oplus x_1 x_2 x_3. \quad \square$$

Algoritmo 1: Obtención de la FNA de una función booleana f a partir de su tabla de verdad

Sean $V_t[i]$ el i-ésimo elemento de la columna V_t, comenzando desde la posición 0, $res(a, b)$ la función que regresa el residuo de dividir a entre b, $bin(b, l)$ una función que nos regresa una cadena de longitud l del número binario b.
Definimos A & A' como la concatenación de A con A'

Entrada: V_t como la columna de salida de la tabla de verdad de la función f, n como el número de variables de la función f.

```
1: bool doXOR
2: String FNA ← null
3: for i = 0 to n − 1 do
4:    doXOR ← true
5:    for  c = 0 to 2^n − 1 do
6:       if res(c, 2^i) = 0 then
7:          doXOR ← not doXOR
8:       end if
9:       if doXOR = true then
10:         V_t[c] ← V_t[c] ⊕ V_t[c − 2^i]
11:      end if
12:   end for
13: end for
14: for  c = 0 to 2^n − 1 do
15:    if V_t[c] = 1 then
16:       String termino ← null
17:       if c > 0 then
18:          String numeroBinario ← bin(2^c, n)
19:          for i = 1 to n do
20:             if numeroBinario[n − i] = 1 then
21:                termino ← termino & x_i
22:             end if
23:          end for
24:       else
25:          termino ← 1
26:       end if
27:       if FNA = null then
28:          FNA ← termino
29:       else
30:          FNA ← FNA & + & termino
31:       end if
32:    end if
33: end for
34: if FNA = null then
35:    FNA ← 0
36: end if
37: return  FNA
```

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	0	1
1	0	0	1
1	1	1	0

Tabla 2.1: Tabla de verdad de la función $f_1(x_1, x_2, x_3)$ del Ejemplo 2.6.1

2.7 Relación entre autómatas y funciones booleanas

En esta sección veremos una generalización de lo que menciona Tao [Tao, 2008] con respecto a las funciones booleanas y los autómatas.

Es importante tener una representación abstracta de los autómatas, con el fin de poder trabajar con máquinas lo suficientemente grandes sin consumir mucho espacio en memoria. Se puede obtener una representación abstracta de un autómata mediante funciones booleanas.

Supongamos que tenemos un autómata con 2^n estados ($n > 0$). Para guardar este autómata en memoria necesitaríamos almacenar 2^n estados y también todas sus transiciones (un total de 2^{n+1}). Suponiendo lo anterior, entonces necesitaríamos guardar 2^n+2^{n+1} unidades de información; por ejemplo, con $n = 5$ necesitamos almacenar 96 unidades de información, lo cual es bastante considerando que es un autómata relativamente pequeño.

Para representar al autómata por medio de funciones booleanas solamente necesitamos almacenar $n + 1$ reglas. Estas reglas corresponden a almacenar la función del valor siguiente de cada una de las n variables más la función de salida del autómata.

2.7.1. Construcción de las funciones booleanas a partir de un autómata finito

Sea $M = \langle \Sigma, \Gamma, S, s_0, \lambda, \delta \rangle$ un autómata, donde $\Sigma = \Gamma = \{0, 1\}$. Para obtener las funciones booleanas en su FNA que representan a M, debemos efectuar los pasos siguientes:

1. Obtener el número de variables para la función booleana, es decir, obtener el valor de n, de la siguiente manera:

$$n = \lceil log_2 \|S\| \rceil$$

donde $\|S\|$ es la cardinalidad del conjunto S.

2. Nombrar los estados. El nombre que le corresponde a cada estado $q_i, i = 0, 1, \ldots, 2^n - 1$, se obtiene de la siguiente forma:

   - Sea i el subíndice del estado q_i. El nombre que le corresponde a q_i es el valor en binario del número i. Si la longitud del valor en binario es menor a n, llenar con 0s a la izquierda hasta que la longitud del nombre sea igual a n.

   Ya que nombramos todos los estados, el i-ésimo bit del nombre del estado corresponde al valor de la i-ésima variable; por ejemplo, supongamos que $n = 3$, el nombre de q_6 es 110, y los valores de las variables son $x_1 = 1, x_2 = 1, x_3 = 0$.

3. Obtener la función booleana correspondiente a la salida del autómata M (ver el Algoritmo 2).

4. Obtener las n fórmulas correspondientes a las variables x_i, para $i = 1, 2, \ldots, n$.

Nota: En cada una de las funciones obtenidas se agrega una variable al principio de la función, esta variable representa el valor del bit de entrada, lo denotamos como x_0.

El Algoritmo 2 muestra cómo obtener las $n+1$ funciones booleanas en su FNA de un autómata M y la Figura 2.7 muestra el diagrama de flujo correspondiente al algoritmo. Como entrada a este diagrama de flujo suponemos que conocemos el número de variables y el autómata $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$, donde $\Sigma = \Gamma = \{0, 1\}$. Con los puntos anteriores obtenemos las $n+1$ fórmulas correspondientes al autómata M.

Ya que obtuvimos las $n+1$ funciones booleanas en su FNA correspondientes al autómata M, la forma de utilizarlas es la siguiente:

Sea $f_i(x_0, x_1, \ldots, x_n)$ la función booleana obtenida para la variable x_i. Sea $s \in S$. Sea $f(x_0, x_1, \ldots, x_n)$ la función booleana en su FNA correspondiente a la salida del autómata M, entonces tenemos:

$$y_0 = f(x_0, x_1, \ldots, x_n),$$
$$x_i' = f_i(x_0, x_1, \ldots, x_n),$$
$$x_i = x_i'.$$

donde x_0 representa el bit de entrada y x_i representa la i-ésima variable del autómata, para $i = 1, 2, \ldots, n$.

De esta forma tenemos un autómata de 2^n estados representado de manera abstracta. Las funciones booleanas $f_i(x_0, x_1, \ldots, x_n)$ representan la función de transición, es decir, $\delta(\langle x_1, x_2, \ldots, x_n \rangle, x_0) = \langle f_1(x_0, x_1, \ldots, x_n), f_2(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n) \rangle$, mientras que la función $f(x_0, x_1, \ldots, x_n)$ representa la función de salida del autómata; esto es, $\lambda(\langle x_1, x_2, \ldots, x_n \rangle, x_0) = f(x_0, x_1, \ldots, x_n)$.

Algoritmo 2: Algoritmo para obtener las $n+1$ funciones booleanas en su FNA que representan a un autómata M dado

- Definimos A & A' como la concatenación de A con A'.
- Sea $TV[]$ un arreglo de tablas de verdad de longitud $n+1$, cada una definida como una tabla hash, donde el *key* representa el vector del estado actual de las variables y *value* representa la salida de la tabla de verdad.
- Sea $s \in S$. Denotamos como s_i al i-ésimo valor del vector de estado siguiente correspondiente a la transición del autómata.
- Sea $TV[i]$.put$(key, value)$ una función para insertar el par $\langle key, value \rangle$ en la i-ésima tabla hash.

Input: El autómata $M = \langle \Sigma, \Gamma, S, s_0, \delta, \lambda \rangle$

Output: Las $n+1$ funciones booleanas que definen al autómata M

1: $n \leftarrow \lceil log_2 \|S\| \rceil$
2: Nombrar todos los estados.
3: **for all** $s \in S$ **do**
4: **for all** $x_0 \in \Sigma$ **do**
5: $vectorX \leftarrow x_0$ & s.nombre
6: $TV[n+1]$.put($vectorX$, $\lambda(s, x_0)$)
7: $s' = \delta(s, x_0)$
8: **for** $i = 1$ **to** n **do**
9: $TV[i]$.put($vectorX$, s'_i)
10: **end for**
11: **end for**
12: **end for**
13: **for** $i = 1$ **to** n **do**
14: Encontrar la FNA correspondiente a la variable x_i de la tabla de verdad $TABVER[i]$ como se vio en la sección anterior.
15: **end for**
16: Encontrar la FNA correspondiente a la función de salida $f(x_0, x_1, \ldots, x_n)$ de la tabla de verdad $TABVER[n+1]$ como se vio en la sección anterior.

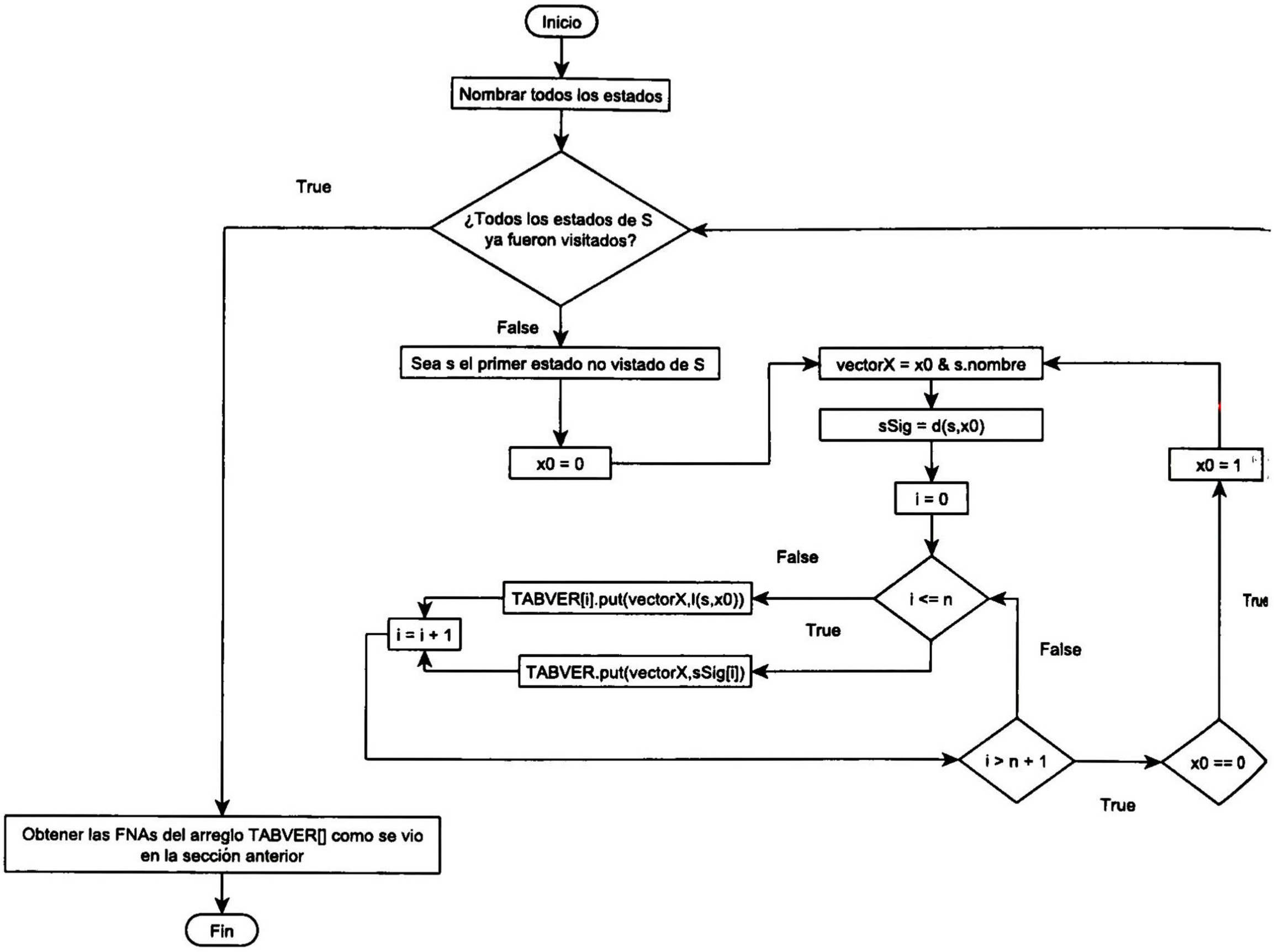


Figura 2.7: Diagrama de flujo para obtener las $n + 1$ funciones booleanas en FNA que corresponden al autómata M

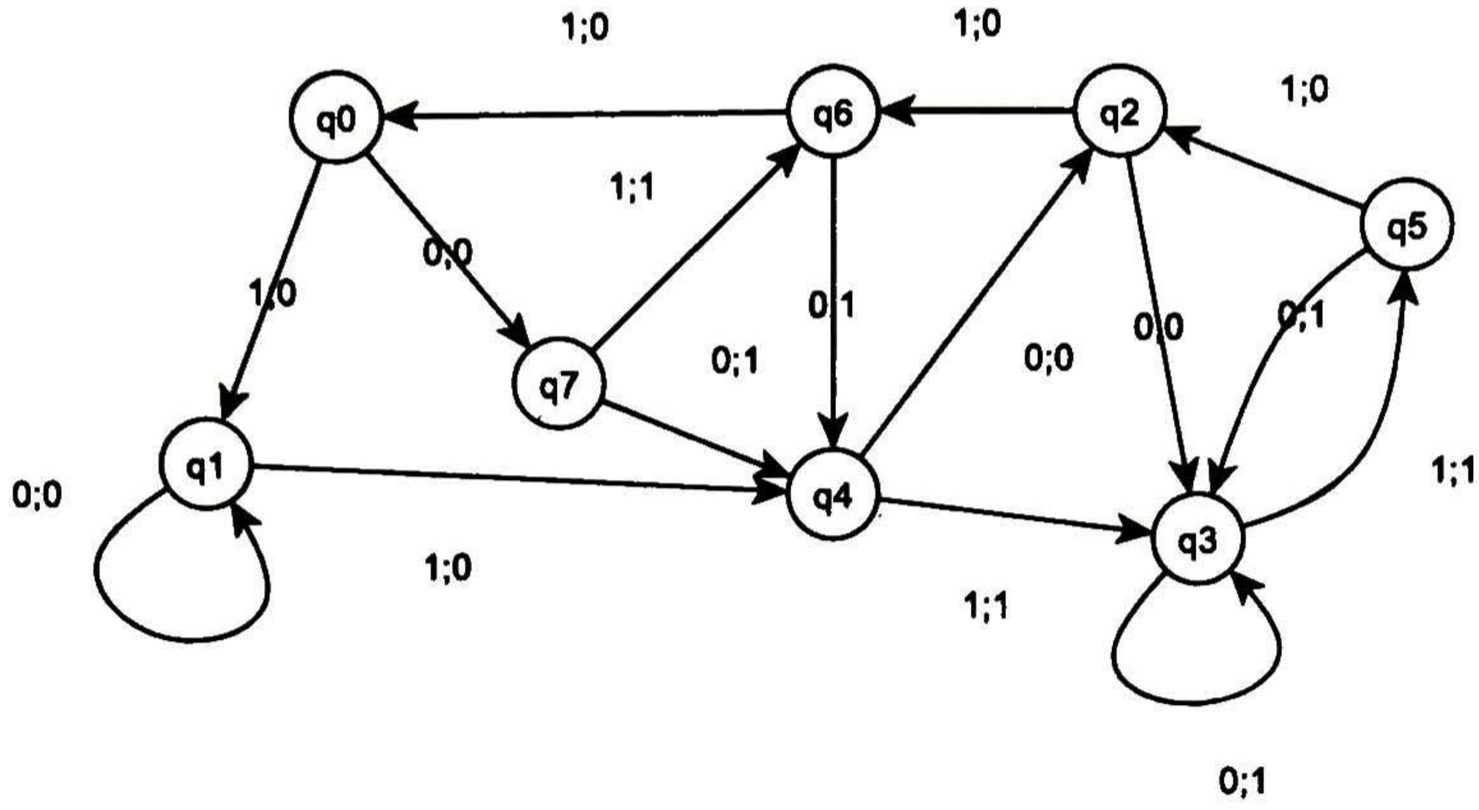


Figura 2.8: Autómata a obtener las fórmulas booleanas que lo representan del ejemplo 2.7.1

Ejemplo 2.7.1 (Obtención de las funciones booleanas correspondientes a un autómata dado). Ilustremos lo mencionado anteriormente con un ejemplo sencillo. Tomemos el autómata de la Figura 2.8. Deseamos encontrar las funciones booleanas que lo representan.

1. El primer paso es obtener el número de variables:

$$n = \lceil log_2 \|S\| \rceil = \lceil log_2 8 \rceil = 3.$$

2. El segundo paso es representar los estados del autómata como vectores binarios, como lo muestra la Tabla 2.2.

3. Después, para cada uno de los estados de M y para cada una de las salidas del autómata obtenemos una tabla de verdad.

Nombre estado	Nuevo nombre
q_0	000
q_1	001
q_2	010
q_3	011
q_4	100
q_5	101
q_6	110
q_7	111

Tabla 2.2: Estados del autómata representados como vectores binarios

$x_0x_1x_2x_3$	y_0
0000	0
0001	0
0010	0
0011	1
0100	0
0101	1
0110	1
0111	1
1000	0
1001	0
1010	0
1011	1
1100	1
1101	0
1110	0
1111	1

$x_0x_1x_2x_3$	x'_1
0000	1
0001	0
0010	0
0011	0
0100	0
0101	0
0110	1
0111	1
1000	0
1001	1
1010	1
1011	1
1100	0
1101	0
1110	0
1111	1

$x_0x_1x_2x_3$	x'_2
0000	1
0001	0
0010	1
0011	1
0100	0
0101	1
0110	0
0111	0
1000	0
1001	0
1010	1
1011	0
1100	1
1101	1
1110	0
1111	1

$x_0x_1x_2x_3$	x'_3
0000	1
0001	1
0010	1
0011	1
0100	0
0101	1
0110	0
0111	0
1000	1
1001	0
1010	0
1011	1
1100	1
1101	0
1110	0
1111	0

4. Ya que tenemos las tablas de verdad y conocemos las propiedades que tienen los poli-

nomios sobre $GF(2)$ aplicamos el Algoritmo 1 para convertir la tabla de verdad de una función booleana a la FNA de la función, obteniendo:

$$y_0 = x_0x_1x_2x_3 + x_0x_1 + x_1x_2 + x_1x_3 + x_2x_3$$

$$x'_1 = x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 + x_0x_1 + x_1x_3 + x_2x_3 + x_0 + x_1 + x_2 + x_3 + 1$$

$$x'_2 = x_0x_2 + x_0x_3 + x_2x_3 + x_0 + x_1 + x_3 + 1$$

$$x'_3 = x_0x_1x_3 + x_1x_2x_3 + x_0x_1 + x_0x_2 + x_0x_3 + x_1x_3 + x_1 + 1. \quad \square$$

2.7.2. Composición de autómatas por medio de su representación en funciones booleanas

Como ya se explicó en secciones anteriores, existen dos tipos de composición de autómatas: (1) Composición en serie y (2) Composición en paralelo. En esta sección mostramos como componer autómatas representados por medio de funciones booleanas.

Sean $\Sigma_1 = \Sigma_2 = \Gamma_1 = \Gamma_2 = \{0,1\}$. Sean $M_1 = \langle \Sigma_1, \Gamma_1, S_1, s_0, \delta_1, \lambda_1 \rangle$ y $M_2 = \langle \Sigma_2, \Gamma_2, S_2, s'_0, \delta_2, \lambda_2 \rangle$, donde $s_0 \in \langle x_0, \ldots, x_n \rangle$, $s'_0 \in \langle x'_0, \ldots, x'_m \rangle$. Sean $f_1(x_0, x_1, \ldots, x_n)$, $\ldots, f_n(x_0, x_1, \ldots, x_n)$ las funciones booleanas que representan la función de transición del autómata M_1. Sean $f'_1(x'_0, x'_1, \ldots, x'_m), \ldots, f'_m(x'_0, x'_1, \ldots, x'_m)$ las funciones booleanas que representan la función de transición del autómata M_2. Sean $f(x_0, x_1, \ldots, x_n)$ y $f'(x'_0, x'_1, \ldots, x'_m)$ las funciones que representan las funciones de salida de los autómatas M_1 y M_2, respectivamente, para $x_0, x_1, \ldots, x_n \in \{0,1\}$ y $x'_0, x'_1, \ldots, x'_m \in \{0,1\}$.

Composición en serie de autómatas representados por funciones booleanas

Sea $C(M_1, M_2) = \langle \Sigma_1, \Gamma_2, S, s, \delta, \lambda \rangle$ el autómata compuesto en serie de M_1 y M_2.

En la representación de autómatas por medio de funciones booleanas sabemos que:

$\delta_1(\langle x_1, \ldots, x_n\rangle, x_0) = \langle f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n)\rangle$

$\lambda_1(\langle x_1, \ldots, x_n\rangle, x_0) = f(x_0, x_1, \ldots, x_n)$, donde $\langle x_1, \ldots, x_n\rangle \in S_1, x_0, \ldots, x_n \in \Sigma_1$

- $\delta_2(\langle x'_1, \ldots, x'_m\rangle, x'_0) = \langle f'_1(x'_0, x'_1, \ldots, x'_m), \ldots, f'_m(x'_0, x'_1, \ldots, x'_m)\rangle$

  $\lambda_2(\langle x'_1, \ldots, x'_m\rangle, x'_0) = f'(x'_0, x'_1, \ldots, x'_m)$, donde $\langle x'_1, \ldots, x'_m\rangle \in S_2,\ x'_0, \ldots, x'_m \in \Sigma_2$

En la composición de autómatas en serie sabemos que:

$$\delta(\langle s_1, s_2\rangle, x_0) = \langle \delta_1(s_1, x_0), \delta_2(s_2, \lambda_1(s_1, x_0))\rangle$$

$$\lambda(\langle s_1, s_2\rangle, x_0) = \lambda_2(s_2, \lambda_1(s_1, x_0))$$

Sabiendo esto y siguiendo la definición de composición en serie, tenemos:

$$\delta(\langle s_1, s_2\rangle, x_0) = \langle f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n),$$
$$f'_1(f(x_0, x_1, \ldots, x_n), x'_1, \ldots, x'_m), \ldots, f'_m(f(x_0, x_1, \ldots, x_n), x'_1, \ldots, x'_m)\rangle$$
$$\lambda(\langle s_1, s_2\rangle, x_0) = f'(f(x_0, x_1, \ldots, x_n), x'_1, \ldots, x'_m)$$
$$s = \langle x_1, \ldots, x_n, x'_1, \ldots, x'_m\rangle$$

Con esto obtenemos las $n+m+1$ funciones booleanas que representan al autómata compuesto $C(M_1, M_2)$:

- Función de salida:

$$f'(f(x_0, x_1, \ldots, x_n), x_1, \ldots, x_m)$$

- Funciones de transición:

$$f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n),$$
$$f'_1(f(x_0, x_1, \ldots, x_n), x'_1, \ldots, x'_m), \ldots, f'_m(f(x_0, x_1, \ldots, x_n), x'_1, \ldots, x'_m)$$

- Estado inicial

$$s = \langle x_1, \ldots, x_n, x'_1, \ldots, x'_m\rangle$$

Composición en paralelo de autómatas representados por funciones booleanas

Sea $P(M_1, M_2) = \langle \Sigma_1 \times \Sigma_2, \Gamma_1 \times \Gamma_2, S_1 \times S_2, s, \delta, \lambda \rangle$ el autómata compuesto en paralelo de M_1 y M_2.

En la representación de autómatas por medio de funciones booleanas sabemos que:

$\delta_1(\langle x_1, \ldots, x_n \rangle, x_0) = \langle f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n) \rangle$

$\lambda_1(\langle x_1, \ldots, x_n \rangle, x_0) = f(x_0, x_1, \ldots, x_n)$, donde $\langle x_1, \ldots, x_n \rangle \in S_1, x_0, \ldots, x_n \in \Sigma$

- $\delta_2(\langle x'_1, \ldots, x'_m \rangle, x'_0) = \langle f'_1(x'_0, x'_1, \ldots, x'_m), \ldots, f'_m(x'_0, x'_1, \ldots, x'_m) \rangle$

  $\lambda_2(\langle x'_1, \ldots, x'_m \rangle, x'_0) = f'(x'_0, x'_1, \ldots, x'_m)$, donde $\langle x'_1, \ldots, x'_m \rangle \in S_2, x'_0, \ldots, x'_m \in \Sigma$

En la composición de autómatas en paralelo sabemos que:

$$\delta(\langle s_1, s_2 \rangle, (x_1, x'_2)) = \langle \delta_1(s_1, x_1), \delta_2(s_2, x'_2) \rangle$$

$$\lambda(\langle s_1, s_2 \rangle, (x_1, x'_2)) = \langle f(x_0, x_1, \ldots, x_n), f'(x'_0, x'_1, \ldots, x'_m) \rangle$$

Sabiendo esto y siguiendo la definición de composición en paralelo, tenemos:

$$\delta(\langle s_1, s_2 \rangle, x_0) = \langle f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n), f'_1(x'_0, x'_1, \ldots, x'_m), \ldots, f'_m(x'_0, x'_1, \ldots, x'_m) \rangle$$

$$\lambda(\langle s_1, s_2 \rangle, x'_0) = \langle f(x_0, x_1, \ldots, x_n), f'(x'_0, x'_1, \ldots, x'_m) \rangle$$

Con esto obtenemos las $n+m+1$ funciones booleanas que representan al autómata compuesto $P(M_1, M_2)$:

- Función de salida:

$$\langle f(x_0, x_1, \ldots, x_n), f'(x'_0, x'_1, \ldots, x'_m) \rangle$$

Funciones de transición:

$$f_1(x_0, x_1, \ldots, x_n), \ldots, f_n(x_0, x_1, \ldots, x_n),$$

$$f'_1(x'_0, x'_1, \ldots, x'_m), \ldots, f'_m(x'_0, x'_1, \ldots, x'_m)$$

- Estado inicial:

$$s = \langle x_1, \ldots, x_n, x'_1, \ldots, x'_m \rangle$$

2.8. Algoritmos para resolver sistemas de ecuaciones polinómicas

El problema de resolver un sistema de ecuaciones polinómicas multivariable sobre un campo finito es *NP-difícil* [Sakalauskas, 2012]. En particular, si el grado de todas las ecuaciones en nuestro sistema es exactamente dos, entonces el problema se le conoce como *MQP* (por sus siglas en inglés, *Multivariate Quadratic Problem*); si el grado de las ecuaciones dentro de nuestro sistema de ecuaciones es mayor a 2, entonces al problema se le conoce como el problema *MPP* (*Multivariate Polinomial Problem*). En este trabajo, nuestros sistemas de ecuaciones polinómicas se encuentran sobre $GF(2)$.

Sean $\mathcal{A}$ un sistema de ecuaciones polinómicas multivariable, n el número de variables en $\mathcal{A}$ y m el número de ecuaciones en $\mathcal{A}$. Decimos que $\mathcal{A}$ está *sobre-determinado* si $\frac{m}{n} \geq 1$; decimos que $\mathcal{A}$ esta *subdeterminado* si $\frac{m}{n} < 1$.

2.8.1. Método de linealización

Para resolver un sistema de ecuaciones polinómicas multivariable por el método de linealización necesitamos realizar los siguientes pasos:

1. Linealizar el sistema de ecuaciones.

2. Resolver el sistema lineal

   2.a) Verificar la solución: si es una solución al sistema de ecuaciones lineales pero no es solución del sistema original (conocida como *solución espuria*) volver al paso 2.

Para resolver un sistema de ecuaciones lineales, normalmente, en criptografía utilizamos la reducción Gaussiana.

Convertir un sistema no lineal a un sistema lineal

Sean n el número de variables en un sistema de ecuaciones no lineal y d el grado mayor dentro del sistema de ecuaciones no lineal. Hay alrededor de $\binom{n}{d}$ monomios de grado menor o igual a d. Consideraremos cada uno de estos términos como una nueva variable V_j, de esta manera obtenemos un sistema lineal del sistema no lineal. Si el sistema lineal está sobre-determinado, entonces podemos utilizar la eliminación Gaussiana sobre el sistema lineal.

Sobre la complejidad de la reducción Gaussiana

En teoría, la reducción Gaussiana tiene una complejidad de a lo más $n^{2.376}$ Sin embargo esto es teoría; en la práctica el algoritmo más rápido que tenemos es el algoritmo de Strassen que tiene una complejidad aproximada de $n^{log_2 7}$. Este tiempo es sólo para encontrar una solución al sistema lineal, que posiblemente sea una solución espuria.

Si denotamos por S el número de todas las posibles soluciones del sistema lineal, entonces las complejidades son:

- **El mejor de los casos**: $\mathcal{O}(n^{log_2 7})$.

- **El peor de los casos:** $\mathcal{O}(Sn^{log_2 7})$.
- **El caso promedio:** $\mathcal{O}\left(\frac{Sn^{log_2 7}}{2}\right)$.

2.8.2. Técnica de relinealización

Debido a que, cuando resolvemos un sistema de ecuaciones no lineales por medio del método de linealización la mayoría de las soluciones que nos entrega no son soluciones para el sistema original, es decir, son soluciones espurias, deseamos añadir restricciones para que, de esa manera, se eliminen algunas soluciones espurias. Las restricciones que añadimos al sistema lineal están basadas en la conmutatividad de la multiplicación del anillo $\mathbb{F}_2[x_1, \ldots, x_n]$ [Kipnis y Shamir, 1999].

Para explicar estas restricciones mostramos un ejemplo. Sea V_{ij} una variable que corresponde al término $x_i x_j$, es decir, $V_{ij} = x_i x_j$. Consideremos 4-tuplas con índices $1 \leq i \leq j \leq k \leq l \leq n$, donde n es el número de variables. Entonces $x_i x_j x_k x_l$ se puede combinar de las siguientes tres maneras:

$$(x_i x_j)(x_k x_l) = (x_i x_k)(x_j x_l) = (x_i x_l)(x_j x_k)$$

así entonces, podemos parametrizar con las siguientes restricciones:

$$V_{ij} V_{kl} = V_{ik} V_{jl} = V_{il} V_{jk}.$$

En total existen $\frac{m^4}{4!}$ maneras diferentes de ordenar una 4-tupla de índices distintos. Así agregamos aproximadamente $\frac{m^4}{12}$ ecuaciones cuadráticas. Después de agregar *algunas* [Kipnis y Shamir, 1999] restricciones aplicamos la técnica de linealización a nuestro nuevo sistema equivalente.

Ejemplo 2.8.1 (Ejemplo de relinealización). Supongamos que tenemos el sistema de ecuaciones cuadraticas con polinomios en $\mathbb{Z}_7[x_1, x_2, x_3]$ siguiente:

$$\begin{aligned}
3x_1x_1 + 5x_1x_2 + 5x_1x_3 + 2x_2x_2 + 6x_2x_3 + 4x_3x_3 &= 5 \\
6x_1x_1 + x_1x_2 + 4x_1x_3 + 4x_2x_2 + 5x_2x_3 + x_3x_3 &= 6 \\
5x_1x_1 + 2x_1x_2 + 6x_1x_3 + 2x_2x_2 + 3x_2x_3 + 2x_3x_3 &= 5 \\
2x_1x_1 + x_1x_3 + 6x_2x_2 + 5x_2x_3 + 5x_3x_3 &= 0 \\
4x_1x_1 + 6x_1x_2 + 2x_1x_3 + 5x_2x_2 + x_2x_3 + 4x_3x_3 &= 0
\end{aligned}$$

reemplazando cada x_ix_j por variables V_{ij} tenemos:

$$\begin{aligned}
3V_{11} + 5V_{12} + 5V_{13} + 2V_{22} + 6V_{23} + 4V_{33} &= 5 \\
6V_{11} + V_{12} + 4V_{13} + 4V_{22} + 5V_{23} + V_{33} &= 6 \\
5V_{11} + 2V_{12} + 6V_{13} + 2V_{22} + 3V_{23} + 2V_{33} &= 5 \\
2V_{11} + V_{13} + 6V_{22} + 5V_{23} + 5V_{33} &= 0 \\
4V_{11} + 6V_{12} + 2V_{13} + 5V_{22} + 1V_{23} + 4V_{33} &= 0
\end{aligned}$$

realizando eliminación Gaussiana:

$$\begin{aligned}
3V_{11} + 5V_{12} + 5V_{13} + 2V_{22} + 6V_{23} + 4V_{33} &= 5 \\
-9V_{12} - 6V_{13} - 7V_{23} - 7V_{33} &= -4 \\
\frac{17}{9}V_{13} - \frac{4}{3}V_{22} - \frac{56}{27}V_{23} + \frac{7}{27}V_{33} &= -\frac{14}{27} \\
\frac{78}{17}V_{22} + \frac{59}{17}V_{23} + \frac{84}{17}V_{33} &= -\frac{32}{17} \\
\frac{829}{78}V_{23} + \frac{6}{13}V_{33} &= -\frac{304}{39}
\end{aligned}$$

reduciendo (*mod* 7):

$$\begin{aligned} 3V_{11} + 5V_{12} + 5V_{13} + 2V_{22} + 6V_{23} + 4V_{33} &= 5 \\ 5V_{12} + V_{13} &= 3 \\ 5V_{13} + V_{22} &= 0 \\ 5V_{22} + V_{23} &= 1 \\ 4V_{23} + V_{33} &= 1 \end{aligned}$$

despejando (*mod* 7):

$$\begin{aligned} 3V_{11} + 5V_{12} + 5V_{13} + 2V_{22} + 6V_{23} + 4V_{33} &= 5 \\ V_{13} &= 3 + 2V_{12} \\ V_{22} &= 2V_{13} \\ V_{23} &= 1 + 2V_{22} \\ V_{33} &= 1 + 3V_{23} \end{aligned}$$

parametrizando (*mod* 7):

$$\begin{aligned} V_{12} &= z \\ 3V_{11} + 5V_{12} + 5V_{13} + 2V_{22} + 6V_{23} + 4V_{33} &= 5 \\ V_{13} &= 3 + 2z \\ V_{22} &= 6 + 4z \\ V_{23} &= 6 + z \\ V_{33} &= 5 + 3z \\ V_1 1 &= 2 + 5z \end{aligned}$$

agregando las restricciones (*técnica de relinealización*):

$$\begin{aligned} V_{11}V_{23} &= V_{12}V_{13} \equiv (2+5z)(6+z) = (z)(3+2z) \equiv 3z^2+z+5=0, \\ V_{12}V_{23} &= V_{13}V_{22} \equiv (z)(6+z) = (3+3z)(6+4z) \equiv 3z+3=0, \\ V_{12}V_{33} &= V_{13}V_{23} \equiv (z)(5+3z) = (3+2z)(6+z) \equiv z^2+4z+3=0 \end{aligned}$$

ahora introducimos dos nuevas variables $z_1 = z, z_2 = z^2$ y las tratamos como variables no relacionadas:

$$\begin{aligned} 3z_2+z_1+5 &= 0, \\ 3z_1+3 &= 0, \\ z_2+4z_1+3 &= 0 \end{aligned}$$

después buscamos la solución de la segunda ecuación y obtenemos $z_1 = 6$, teniendo esto en cuenta tenemos $z_2 = 1$.

Ahora, trabajando hacia atrás (*mod* 7):

$$\begin{aligned} V_{22} &= 6+4(6) \equiv 2 \\ V_{33} &= 5+3z \equiv 2 \\ V_{11} &= 2+5(6) \equiv 4 \end{aligned}$$

encontrando las raíces tenemos (*mod* 7):

$$\begin{aligned} x_1 &= \pm 2 \\ x_2 &= \pm 3 \\ x_3 &= \pm 3 \end{aligned}$$

utilizando los valores $y_{12} = 6, y_{23} = 5$, tenemos dos posibles soluciones al sistema inicial:

- $x_1 = 2, x_2 = 3, x_3 = 4$.

- $x_1 = 5, x_2 = 4, x_3 = 3.$

las cuales no son soluciones espurias. □

2.8.3. Algoritmo XL

Como mencionamos en la sección anterior, Kipnis y Shamir [Kipnis y Shamir, 1999] presentaron un nuevo método para resolver sistemas de ecuaciones sobre-definidas, llamado *relinealización.* La idea básica es agregar a un sistema de ecuaciones lineal dado un número adicional de ecuaciones, las cuales expresan las relaciones que existen entre las variables; la técnica está basada en la conmutatividad de la multiplicación. El algoritmo XL (de las siglas en inglés, *eXtended Linearizations*) puede ser visto como una mejora a la técnica de relinealización. Este algoritmo fue presentado por Courtois et al. [Courtois, 2003] y la idea principal es agregar restricciones de una manera más algorítmica (más que intuitiva, como en la técnica de relinealización).

A continuación daremos una breve explicación de este algoritmo. Para un estudio más completo invitamos al lector a consultar las referencias de esta sección.

Sean $\mathcal{K}$ un campo, m el número de ecuaciones y $\mathcal{A}$ un sistema de ecuaciones polinómicas multivariable de la forma $l_k = 0$ $(1 \leq k \leq m)$, donde cada l_k es un polinomio multivariable de la forma $f_k(x_1, x_2, \ldots, x_n) - b_k$.

El problema es encontrar al menos una solución $x = (x_1, x_2, \ldots, x_n) \in \mathcal{K}^n$ para algunos bits dados $b = (b_1, b_2, \ldots, b_m) \in \mathcal{K}^m$

Denotamos a todos los polinomios de la forma $\prod_{j=1}^{k} x_{ij} * l_i$ como $\mathcal{I}_k$.

Sea $D \in \mathbb{N}$. Consideraremos todos los polinomios de grado menor o igual a D.

Definición del algoritmo XL

Para realizar el algoritmo XL, se deben ejecutar los siguientes pasos:

1. **Multiplicar**: Generar todos los productos $\prod_{j=1}^{k} x_{ij} * l_i \in \mathcal{I}_{D-2}$ para $k \leq D-2$. Es decir, multiplicaremos todas las ecuaciones en $\mathcal{A}$ por todos los términos de grado menor o igual a $D-2$.

2. **Linealizar**: Consideraremos cada monomio, de grado menor o igual a D, que aparezca en las ecuaciones obtenidas en el paso 1, e introduciremos una variable fresca por cada monomio considerado, después realizaremos eliminación Gaussiana sobre las ecuaciones linealizadas.

3. **Resolver**: Asumimos que el paso 2 nos lleva a al menos una ecuación de una sola variable de x. Resolvemos esta ecuación sobre el campo finito.

4. **Repetir**: Simplificamos las ecuaciones, con el(los) valor(es) obtenido(s) en el paso anterior y repetimos el proceso para encontrar los valores de las otras variables.

El algoritmo XL es bastante sencillo, pero no está muy claro para qué valores de n y m termina. A pesar de que el algoritmo XL no es muy extenso, su complejidad no ha sido determinada.

2.9. Conversión de un sistema de ecuaciones polinómicas a FNC

Una instancia del problema en la forma normal conjuntiva SAT (*FNC-SAT*), también conocido como *CNF-SAT* por sus siglas en inglés, es un conjunto de cláusulas, donde

cada cláusula es una disyunción de varias literales. Si un conjunto de valores, para todas las n literales, hace que la valuación de todas las cláusulas sea verdadera, entonces se dice que es una asignación satisfacible. De esta forma, el conjunto de cláusulas se puede expresar como una larga expresión lógica, llamada conjunción de todas las cláusulas.

Para mayor información sobre la conversión (que se expondrá brevemente a continuación) de un sistema de ecuaciones polinómicas multivariable a FNC se puede consultar [Bard et al., 2007].

La conversión se basa en tres pasos principales [Bard et al., 2007]: (1) el sistema de polinomios original se convierte a un (gran) sistema lineal; (2) se realiza un pre-procesamiento para hacer el sistema más manejable a la conversión, (3) el sistema lineal se convierte a un conjunto de cláusulas equivalente.

Nota. Ya que la FNC no tiene ninguna constante, cambiaremos en donde aparezca la constante por la literal T, es decir, si la ecuación que estamos convirtiendo a FNC contiene la constante igual a 1, agregamos la variable T.

Paso 1: De un sistema polinomial a un sistema lineal

Cada nuevo monomio que aparezca en el sistema polinomial lo sustituimos con una variable fresca V_j, donde $j \in \mathbb{N}$. También, cada presencia de la constante 1 se reemplaza con la variable especial T.

Después de esto, cada polinomio es ahora una suma de términos de grado uno.

Paso 2: Pre-proceso del sistema lineal

Solamente utilizaremos ecuaciones homogéneas, es decir, $V_i + V_j + V_k + \ldots + a_1 T = 0$ ($a_1 \in \{0, 1\}$). Si existe una ecuación no homogénea, simplemente agregamos la variable especial T y reemplazamos el lado derecho de la ecuación por 0, así obtenemos un sistema

de ecuaciones equivalente de la forma $V_i + V_j + V_k + \ldots + T = 0$. Después de eso cada polinomio que tenemos es ahora una suma de variables, o equivalentemente un XOR lógico. Por desgracia, es conocido que el manejo de grandes XORs es un problema difícil para los SAT-Solvers [Creignou y Daude, 1999].

En general, para convertir un polinomio lineal de longitud ℓ a su FNC es necesario poner todas las posibles combinaciones con todos los números impares de negaciones menores o iguales a ℓ. En particular, la suma de términos $V_i + V_j + V_k = 0$ es equivalente a la fórmula proposicional:

$$(\overline{V_i} \vee V_j \vee V_k) \wedge (V_i \vee \overline{V_j} \vee V_k) \wedge (V_i \vee V_j \vee \overline{V_k}) \wedge (\overline{V_i} \vee \overline{V_j} \vee \overline{V_k}).$$

Sean $\ell \in \mathbb{N}$, $i = 2\left\lfloor \frac{\ell}{2} \right\rfloor$ y $P(x) = V_i + V_j + V_k + \ldots + V_\ell = 0$ un polinomio lineal. Para convertir $P(x)$ en su FNC, necesitamos

$$\binom{\ell}{1} + \binom{\ell}{3} + \binom{\ell}{5} + \ldots + \binom{\ell}{i} = 2^{\ell-1}$$

cláusulas, lo cual es exponencial. Para remediar esto, debemos *cortar* cada polinomio en polinomios equivalentes más pequeños de longitud menor a ℓ. A la longitud más pequeña que ℓ se le conoce como *número de corte* [Bard et al., 2007].

Ejemplo 2.9.1 (Ejemplo del *número de corte* $= 4$)**.** Sea $V_1 + V_2 + V_3 + \ldots + V_\ell = 0$ una ecuación lineal homogénea de longitud ℓ. Esta ecuación es equivalente al sistema de ecuaciones lineales:

$$V_1 + V_2 + V_3 + y_1 = 0$$

$$y_1 + V_4 + V_5 + y_2 = 0$$

$$\vdots$$

$$y_i + V_{2i+2} + V_{2i+3} + y_{i+1} = 0$$

$$\vdots$$

$$y_h + V_{\ell-2} + V_{\ell-1} + V_\ell = 0$$

si ℓ es par, donde cada y_i es una variable auxiliar, para $i = 1, 2, \ldots, h$. Si ℓ es impar, la última ecuación será una suma de tres términos.

Podemos calcular h como sigue:

$$h = \left\lceil \frac{\ell}{2} - 2 \right\rceil$$

Así, cuando convertimos una ecuación lineal homogénea de longitud ℓ a su FNC, tendremos $h + 1$ subsumas, cada una de las cuales tendrá ocho cláusulas de longitud cuatro. Es decir, tendremos $2^5 h$ cláusulas por cada polinomio en nuestro sistema de ecuaciones, en vez de $2^{\ell-1}$ cláusulas. □

Paso 3: De un sistema lineal a la forma normal conjuntiva

Una vez realizados los pasos 1 y 2, cada polinomio es una suma de variables de longitud igual o menor al *número de corte*. En general, cuando se desea convertir una ecuación lineal a su forma normal conjuntiva, se agregan todas las cláusulas cuyo número de negaciones sea impar y menor al *número de corte*.

En particular, si el *número de corte* es 4, entonces la suma $V_i + V_j + V_k + V_l = 0$ es equivalente a la fórmula proposicional:

$$\begin{aligned}&(\overline{V_i} \vee V_j \vee V_k \wedge V_l) \wedge (V_i \vee \overline{V_j} \vee V_k \wedge V_l) \wedge (V_i \vee V_j \vee \overline{V_k} \wedge V_l) \wedge (V_i \vee V_j \vee V_k \wedge \overline{V_l}) \wedge \\ &(\overline{V_i} \vee \overline{V_j} \vee \overline{V_k} + V_l) \wedge (\overline{V_i} \vee \overline{V_j} \vee V_k + \overline{V_l}) \wedge (\overline{V_i} \vee V_j \vee \overline{V_k} + \overline{V_l}) \wedge (V_i \vee \overline{V_j} \vee \overline{V_k} + \overline{V_l}),\end{aligned} \tag{2.9}$$

Ya que todos los polinomios en nuestro sistema de ecuaciones polinómicas equivalente, que creamos con los pasos 1 y 2, son sumas de términos de longitud 4 (o 3, si ℓ es impar), aplicamos una instancia de la fórmula proposicional 2.9 y de esta forma convertimos el sistema de ecuaciones polinómicas original en su FNC.

Ejemplo 2.9.2 (Conversión de una ecuación polinómica a su FNC). Supongamos que tenemos la ecuación polinómica homogénea $x_1x_2x_3 + x_2x_4 + x_3x_5 + x_1 + x_2 + x_3 + 1 = 0$.

1. Linealizamos el sistema como sigue:

$$\begin{aligned} V_1 &= x_1x_2x_3 \\ V_2 &= x_2x_4 \\ V_3 &= x_3x_5 \\ V_4 &= x_1 \\ V_5 &= x_2 \\ V_6 &= x_3 \\ V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + T &= 0 \end{aligned}$$

2. *Cortamos la ecuación*:

$$\begin{aligned} V_1 + V_2 + V_3 + y_1 &= 0 \\ y_1 + V_4 + V_5 + y_2 &= 0 \\ y_2 + V_6 + T &= 0 \end{aligned}$$

Convertimos el sistema de ecuaciones lineal homogéneo a su FNC:

La ecuación $V_1 + V_2 + V_3 + y_1 = 0$ es equivalente a:

$$(\overline{V_1} \vee V_2 \vee V_3 \wedge y_1) \wedge (V_1 \vee \overline{V_2} \vee V_3 \wedge y_1) \wedge (V_1 \vee V_2 \vee \overline{V_3} \wedge y_1) \wedge (V_1 \vee V_2 \vee V_3 \wedge \overline{y_1}) \wedge$$
$$(\overline{V_1} \vee \overline{V_2} \vee \overline{V_3} + y_1) \wedge (\overline{V_1} \vee \overline{V_2} \vee V_3 + \overline{y_1}) \wedge (\overline{V_1} \vee V_2 \vee \overline{V_3} + \overline{y_1}) \wedge (V_1 \vee \overline{V_2} \vee \overline{V_3} + \overline{y_1}).$$

La ecuación $y_1 + V_4 + V_5 + y_2 = 0$ es equivalente a:

$$(\overline{y_1} \vee V_4 \vee V_5 \wedge y_2) \wedge (y_1 \vee \overline{V_4} \vee V_5 \wedge y_2) \wedge (y_1 \vee V_4 \vee \overline{V_5} \wedge y_2) \wedge (y_1 \vee V_4 \vee V_5 \wedge \overline{y_2}) \wedge$$
$$(\overline{y_1} \vee \overline{V_4} \vee \overline{V_5} + y_2) \wedge (\overline{y_1} \vee \overline{V_4} \vee V_5 + \overline{y_2}) \wedge (\overline{y_1} \vee V_4 \vee \overline{V_5} + \overline{y_2}) \wedge (y_1 \vee \overline{V_4} \vee \overline{V_5} + \overline{y_2}).$$

La ecuación $y_2 + V_6 + T = 0$ es equivalente a:

$$(\overline{y_2} \vee V_6 \vee T) \wedge (y_2 \vee \overline{V_6} \vee T) \wedge (y_2 \vee V_6 \vee \overline{T}) \wedge (\overline{y_2} \vee \overline{V_6} \vee \overline{T}).$$

De esta manera, obtenemos la FNC del sistema lineal.

Capítulo 3

Criptoanálisis lógico-algebraico

En este capítulo nos enfocamos a exponer nuestra propuesta. Comenzamos definiendo la técnica *RECNF*, la cual nos permite agregar información adicional a la FNC sobre el sistema original. Después mostramos un criptoanálisis que nombramos *criptoanálisis lógico-algebraico*. Posteriormente mostramos el procedimiento para criptoanalizar cualquier cifrado que se pueda modelar mediante un sistema de ecuaciones polinómicas.

3.1. La técnica RECNF

En la sección 2.8 mostramos 3 técnicas diferentes para resolver sistemas de ecuaciones polinómicas multivariable: (1) el método por linealización, (2) la técnica de relinealización y (3) el algoritmo XL; todas las técnicas antes mencionadas transforman el sistema de ecuaciones polinómicas en un sistema de ecuaciones lineales equivalente, pero al realizar esta transformación pierden cierta información del sistema original.

Esta pérdida se debe a que en el sistema original sabemos que los términos deben cumplir ciertas propiedades con respecto a la multiplicación, pero en el sistema lineal perde-

mos esta información debido a que introducimos una variable fresca cada vez que encontramos un término de grado mayor a 1.

Entre más información sobre el sistema original agreguemos al sistema lineal menos soluciones *espurias* obtendremos (las soluciones espurias son aquellas soluciones que satisfacen al sistema lineal, pero no satisfacen al sistema original), ya que, si agregamos alguna restricción al sistema lineal, la solución que satisface al sistema lineal pero no al sistema original ya no satisfará al sistema generado.

Para poder utilizar esta técnica, debemos primero haber convertido el sistema polinomial multivariable en su FNC, como se vio en la sección 2.9.

3.1.1. Descripción de la técnica RECNF

La idea general de la técnica RECNF es agregar información que se pierde cuando linealizamos un sistema no lineal. Esta información la agregamos por medio de cláusulas que necesitamos agregar a la forma normal conjuntiva del sistema polinomial multivariable que convertimos.

Primero debemos agregar la cláusula (T) a la forma normal conjuntiva, después agregamos la información adicional. La información asociada al sistema original que no tiene el sistema lineal la obtenemos de la siguiente manera:

Supongamos que tenemos una variable lineal que puede ser representada como el producto de dos o más variables del sistema de ecuaciones lineales (asumiendo que hay m variables en total), es decir,

$$V_i = \underbrace{V_j V_k \dots V_z}_{m} \tag{3.1}$$

con $j < k < \ldots < z$.

Esta ecuación puede ser representada como una proposición de la forma:

$$V_i \iff (V_j \wedge V_k \wedge \ldots \wedge V_z)$$

que es lógicamente equivalente a:

$$(\overline{V_i} \vee V_j) \wedge (\overline{V_i} \vee V_k) \wedge \ldots \wedge (\overline{V_i} \vee V_z) \wedge (V_i \vee \overline{V_j} \vee \overline{V_k} \vee \ldots \vee \overline{V_z}) \tag{3.2}$$

Ahora tenemos $m + 1$ nuevas cláusulas de tamaño dos y una cláusula de tamaño $m + 1$.

Una vez que obtenemos estas nuevas cláusulas, debemos agregarlas, junto con la cláusula (T), a la forma normal conjuntiva del sistema original.

Así entonces, si tenemos una variable lineal como en la fórmula 3.1 podemos obtener su forma normal conjuntiva directamente, ya que la fórmula 3.2 nos da la fórmula general para cualquier descomposición que encontremos. Una vez que obtuvimos todas las descomposiciones posibles de una variable lineal lo único que nos falta es agregar las cláusulas generadas a la forma normal conjuntiva del sistema lineal.

La Figura 3.1 muestra el diagrama de flujo general para utilizar la técnica RECNF.

Observación importante: Si todos los monomios de grado 1 aparecen, en las nuevas restricciones, agregando esta información (incluyendo la cláusula T) a la forma normal conjuntiva, se eliminan todas las soluciones que satisfacen al sistema lineal pero no satisfacen al sistema original.

Ejemplo 3.1.1 (Técnica RECNF). Supongamos que $V_1 = x_1x_2x_3x_5, V_2 = x_1x_2, V_3 = x_3x_5,$ $V_4 = x_3$ y $V_5 = x_5$

Aplicando la técnica RECNF obtenemos lo siguiente:

$$V_1 = V_2 V_3$$

$$V_1 = V_2 V_4 V_5$$

$$V_3 = V_4 V_5$$

Entonces, las nuevas cláusulas que necesitamos agregar a la forma normal conjuntiva son:

$$(T)$$

$$(\overline{V_1} \vee V_2) \wedge (\overline{V_1} \vee V_3) \wedge (V_1 \vee \overline{V_2} \vee \overline{V_3})$$

$$(\overline{V_1} \vee V_2) \wedge (\overline{V_1} \vee V_4) \wedge (\overline{V_1} \vee V_5) \wedge (V_1 \vee \overline{V_2} \vee \overline{V_4} \vee \overline{V_5})$$

$$(\overline{V_3} \vee V_4) \wedge (\overline{V_3} \vee V_5) \wedge (V_3 \vee \overline{V_4} \vee \overline{V_5}) \quad \square$$

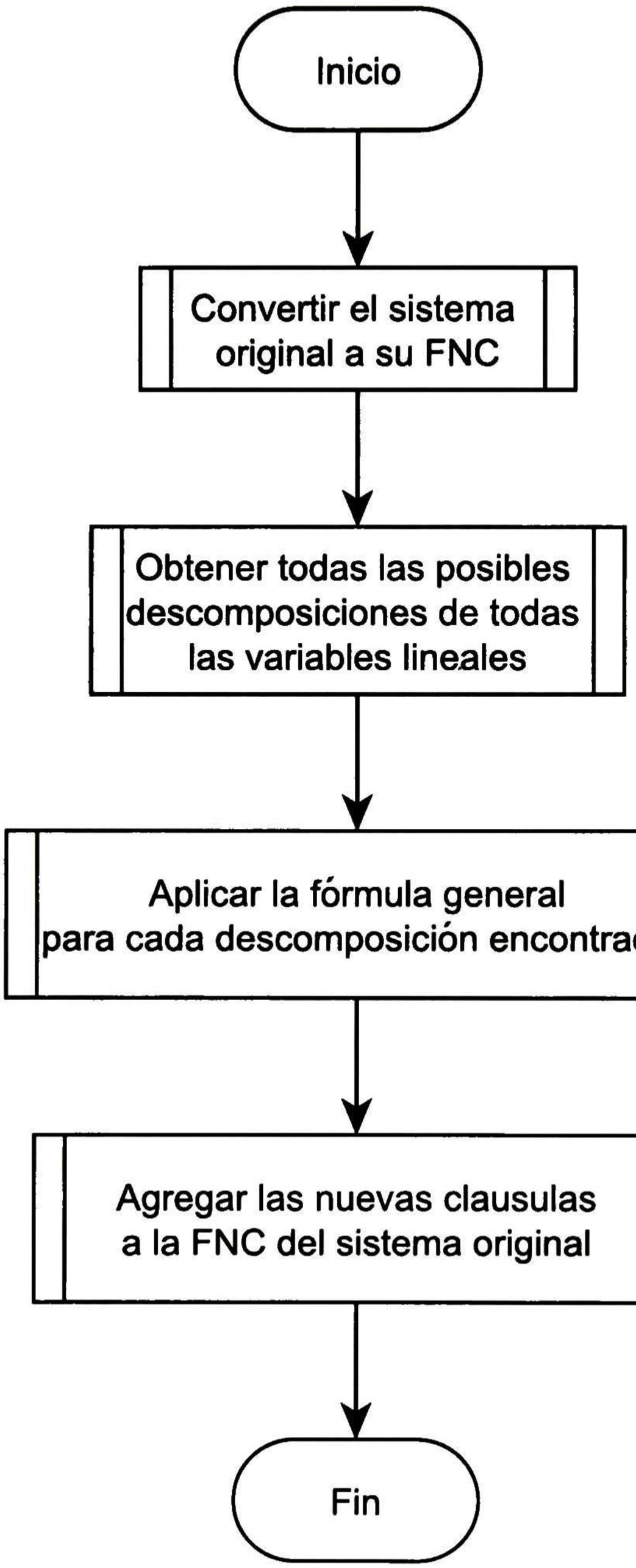


Figura 3.1: Diagrama de flujo de la técnica RECNF

3.1.2. Complejidad de la técnica RECNF

La complejidad mayor de esta técnica se presenta al encontrar las posibles descomposiciones. El encontrar todas las posibles descomposiciones de una variable lineal de longitud l tiene un costo de:

$$\sum_{i=1}^{l-1} \binom{l}{i} = 2^l - 2$$

Por lo tanto, la complejidad es:

$$\mathcal{O}(2^l - 2)$$

3.2. Criptoanálisis lógico-algebraico

En la Sección 3.1 mostramos cómo obtener información adicional y agregar esa información a la forma normal conjuntiva del sistema original. Nos enfocaremos en la manera de utilizar esta información para criptoanalizar cualquier cifrado que se pueda modelar por medio de un sistema de ecuaciones.

Este criptoanálisis se basa, como ya habíamos mencionado, en dos tipos de criptoanálisis: (1) el criptoanálisis lógico y (2) el criptoanálisis algebraico. Del criptoanálisis algebraico utilizamos la idea de modelar el cifrado como un sistema de ecuaciones y del criptoanálisis lógico utilizamos el procedimiento para convertir las ecuaciones dadas en su forma normal conjuntiva y utilizar los SAT-Solvers para obtener un modelo que satisfaga a la forma normal conjuntiva.

Una vez que tenemos un modelo que satisface a nuestra forma normal conjuntiva obtenemos la respuesta de ese modelo y eliminamos las soluciones espurias. Cuando encontramos una solución espuria, detenemos el criptoanálisis, ya que hemos obtenido la clave

privada, es decir, hemos logrado romper el criptosistema.

El criptoanálisis lógico-algebraico consiste básicamente en cinco pasos:

1. Obtener suficientes ecuaciones para que el sistema sea *sobre-determinado*, es decir, que tengamos un número de ecuaciones mayor o igual que de incógnitas.

2. Aplicar el paso 1 en el algoritmo XL (Sección 2.8.3) con $D = 3$, con las ecuaciones obtenidas en el paso 1. Es decir, vamos a multiplicar cada una de las variables que tenemos en nuestro sistema de ecuaciones polinómicas por todas las ecuaciones en el. Una vez que realizamos estas multiplicaciones tendremos nm ecuaciones, donde n es el número de variables y m es el número de ecuaciones dadas.

3. Convertir el sistema de ecuaciones obtenido en el paso 2 a su forma normal conjuntiva (Sección 2.9).

4. Utilizar la técnica RECNF, vista en la Sección 3.1, y agregar las cláusulas generadas a la forma normal conjuntiva obtenida en el paso 3. Después utilizar el SAT-Solver para obtener una respuesta.

5. Obtenemos la respuesta al sistema original del modelo que nos entrega el SAT-Solver. Si todas las variables ya tienen un valor, el criptosistema se ha roto, ya que la solución encontrada corresponde a la clave privada. Pero si existe al menos una variable indeterminada, entonces:

   - Generamos nuevas ecuaciones, partiendo del sistema original; esto lo hacemos sustituyendo los valores obtenidos en las ecuaciones originales. Una vez que hacemos esto, volvemos al paso 2 con las ecuaciones generadas.

La Figura 3.2 muestra el diagrama de flujo del criptoanálisis lógico-algebraico.

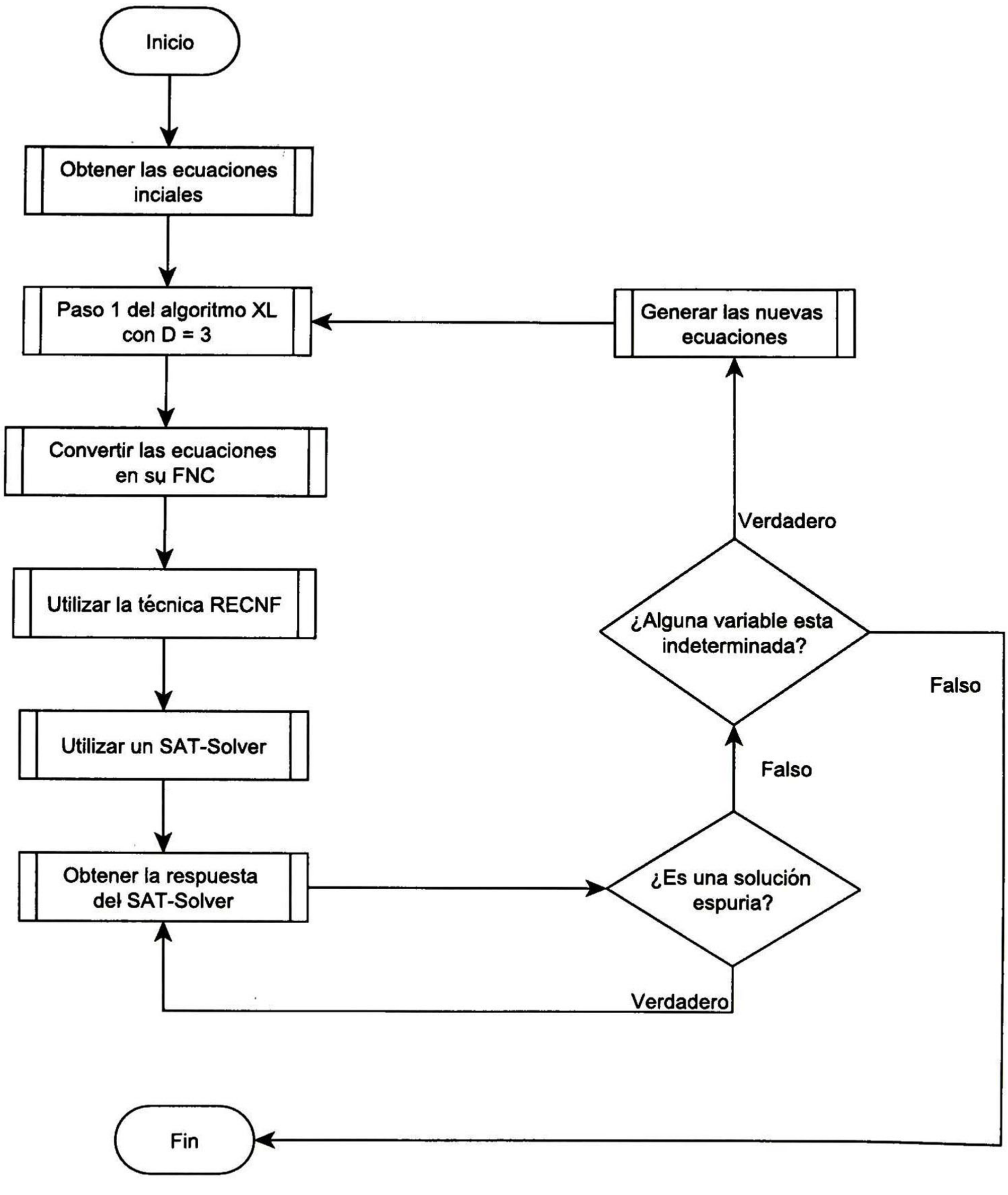


Figura 3.2: Diagrama de flujo del criptoanálisis lógico-algebraico

3.2.1. Complejidad del criptoanálisis lógico-algebraico

La complejidad para obtener el sistema de ecuaciones polinómicas *sobre-determinado* en el paso 1 es lineal, ya que a cada paso de tiempo obtendremos una ecuación. En otras palabras, supongamos que queremos obtener m ecuaciones, la complejidad para obtener las m ecuaciones será de $\mathcal{O}(m)$.

Una vez que obtenemos las ecuaciones iniciales debemos multiplicar cada una de las n variables por todas las m ecuaciones iniciales. Entonces, ya que debemos realizar n veces m multiplicaciones, la complejidad de este paso es $\mathcal{O}(nm)$.

Para convertir las ecuaciones polinómicas en su forma normal conjuntiva primero *cortamos* las ecuaciones. Sea ℓ el máximo número de términos en una ecuación dentro de nuestras m ecuaciones. Sea c el número de corte. Entonces tenemos:

$$\mathcal{O}\left(\left\lceil \frac{\ell - 2}{c} \right\rceil\right)$$

Una vez que cortamos nuestras ecuaciones lo siguiente es convertir las nuevas ecuaciones obtenidas en su forma normal conjuntiva. Sea m_1 el número de ecuaciones obtenidas en el paso anterior. La complejidad de la transformación es:

$$\mathcal{O}(m_1)$$

La complejidad de la técnica RECNF, como lo mostramos en la Subsección 3.1.2, es $\mathcal{O}(2^l - 2)$, donde l es el máximo número de variables que puede tener un término en el sistema de ecuaciones.

Después de utilizar la técnica RECNF empleamos el SAT-Solver para que nos genere un modelo que satisfaga a la forma normal conjuntiva. Este paso es el más difícil de todos,

debido a que el SAT-Solver comienza de manera aleatoria con una semilla (*Seed*). Ya que utilizamos el SAT-Solver como una caja negra, es más difícil aún obtener la complejidad de este paso. Asumiremos que $\mathcal{O}(S)$ es la complejidad del SAT-Solver.

La complejidad total del criptoanálisis lógico-algebraico es:

$$\mathcal{O}(m) + \mathcal{O}(nm) + \mathcal{O}\left(\left\lceil \frac{\ell - 2}{c} \right\rceil\right) + \mathcal{O}(m_1) + \mathcal{O}(2^l - 2) + \mathcal{O}(S) \tag{3.3}$$

La complejidad mayor en la fórmula anterior es la del SAT-Solver. Es decir, la complejidad final, del criptoanálisis lógico-algebraico es:

$$\mathcal{O}(S) \tag{3.4}$$

Capítulo 4

Casos de estudio

Como vimos en el capítulo anterior, la complejidad del criptoanálisis lógico-algebraico se basa en la complejidad del SAT-Solver. Ya que utilizamos al SAT-Solver como una caja negra, es muy difícil saber la complejidad de este ataque; es por eso que realizamos diferentes instancias del criptoanálisis a los criptosistemas y, con ayuda de la estadística, obtenemos un tiempo esperado de ejecución del criptoanálisis para cada criptosistema en particular.

En este capítulo detallaremos la herramienta implementada para el criptoanálisis lógico-algebraico; después daremos los datos sobre el equipo en el cual las pruebas fueron ejecutadas; posteriormente describiremos el criptosistema Trivium y sus variantes (Bivium-A y Bivium-B); después mostramos los resultados obtenidos del criptoanálisis a Bivium-B, luego describimos de una manera sucinta el criptosistema FAPKC3; por último mostramos los resultados obtenidos a una instancia de FAPKC3.

En este trabajo utilizamos el *número de corte* $c = 4$; este número nos sirve en la conversión de un sistema de ecuaciones polinómicas a su forma normal conjuntiva.

4.1. Detalles sobre la herramienta creada para el criptoanálisis

Desarrollamos una herramienta de software la cual nos ayuda con el criptoanálisis de los criptosistemas implementados, esta herramienta fue desarrollada en Java. Cuenta con una implementación del criptoanálisis lógico-algebraico con el *número de corte* $c = 4$. Además, la herramienta es capaz de generar un sistema de ecuaciones para Trivium, Bivium-B y Bivium-A, así como una instancia pequeña del FAPKC3. También es capaz de generar un sistema de ecuaciones aleatorio, en base a las especificaciones, para el criptosistema que se le indique, ya sea Bivium-A, Bivium-B, Trivium o FAPKC3.

Para utilizar el criptoanálisis lógico-algebraico se debe indicar la ruta en donde se encuentra el archivo con el sistema de ecuaciones. El archivo con el sistema de ecuaciones debe tener un formato especifico; este formato es muy sencillo: todas las variables tienen que ser de la forma x_i, donde x es fija, donde $0 < i \leq 2^{32} - 1$, xi es la referencia para la i-ésima variable.

Como SAT-Solver utilizamos SAT4j [Le Berre et al., 2010]. Este es un SAT-Solver basado en MiniSAT [Eén y Sörensson, 2004], la diferencia es que SAT4j está desarrollado en Java.

4.2. Datos del equipo sobre el que se ejecutaron las pruebas

Las pruebas se corrieron en una computadora con 14 GB de memoria RAM y un procesador i7 @2.8 GHz.

4.3. Trivium

El proyecto eSTREAM comenzó en octubre del 2004, fue lanzado para encontrar un cifrado de flujo, después que el proyecto NESSIE terminó en 2003 sin recomendar ningún cifrado de flujo. Comenzó con una convocatoria para el proyecto, después en cada fase se fueron descartando los candidatos débiles, es decir, aquellos candidatos que no superaron los criptoanálisis. Todos los candidatos de eSTREAM se encuentran divididos en dos categorías: cifrados orientados a hardware y cifrados orientados a software. Trivium es un cifrado de flujo orientado a hardware y fue introducido en [De Cannière, 2006]. Hasta ahora ningún ataque aplicado a Trivium ha tenido éxito, es decir no ha sido posible un ataque contra Trivium que corra más rápido que la búsqueda exhaustiva.

Raddum [Raddum, 2006] presentó dos versiones reducidas de Trivium llamadas Bivium-A y Bivium-B. La intención de estas versiones es encontrar ataques hacia Bivium-A o Bivium-B y después extenderlos a Trivium.

4.3.1. Descripción de Trivium

Trivium puede ser descrito como un autómata finito cuyo estado inicial se deriva de la clave secreta y un vector de inicialización (IV). Después de poner la clave secreta y el vector de inicialización se realizan algunas transiciones, las cuales mezclan el estado interno, a esta fase se le conoce como la fase de inicialización. Después de la fase de inicialización se comienzan a producir la secuencia de salida (también conocida como *keystream*). Trivium y Bivium producen un bit de salida por cada paso de tiempo (clock) del cifrado. Para poder encriptar un mensaje, digamos el mensaje m, se suma en módulo 2 (XOR) el i-ésimo el bit de la salida con el i-ésimo bit del mensaje, es decir, $m_i \oplus z_i = c_i, i = 0, 1, 2, \ldots$, donde m_i es el i-ésimo bit del mensaje m, z_i es el i-ésimo bit del keystream y c_i es el i-ésimo bit del

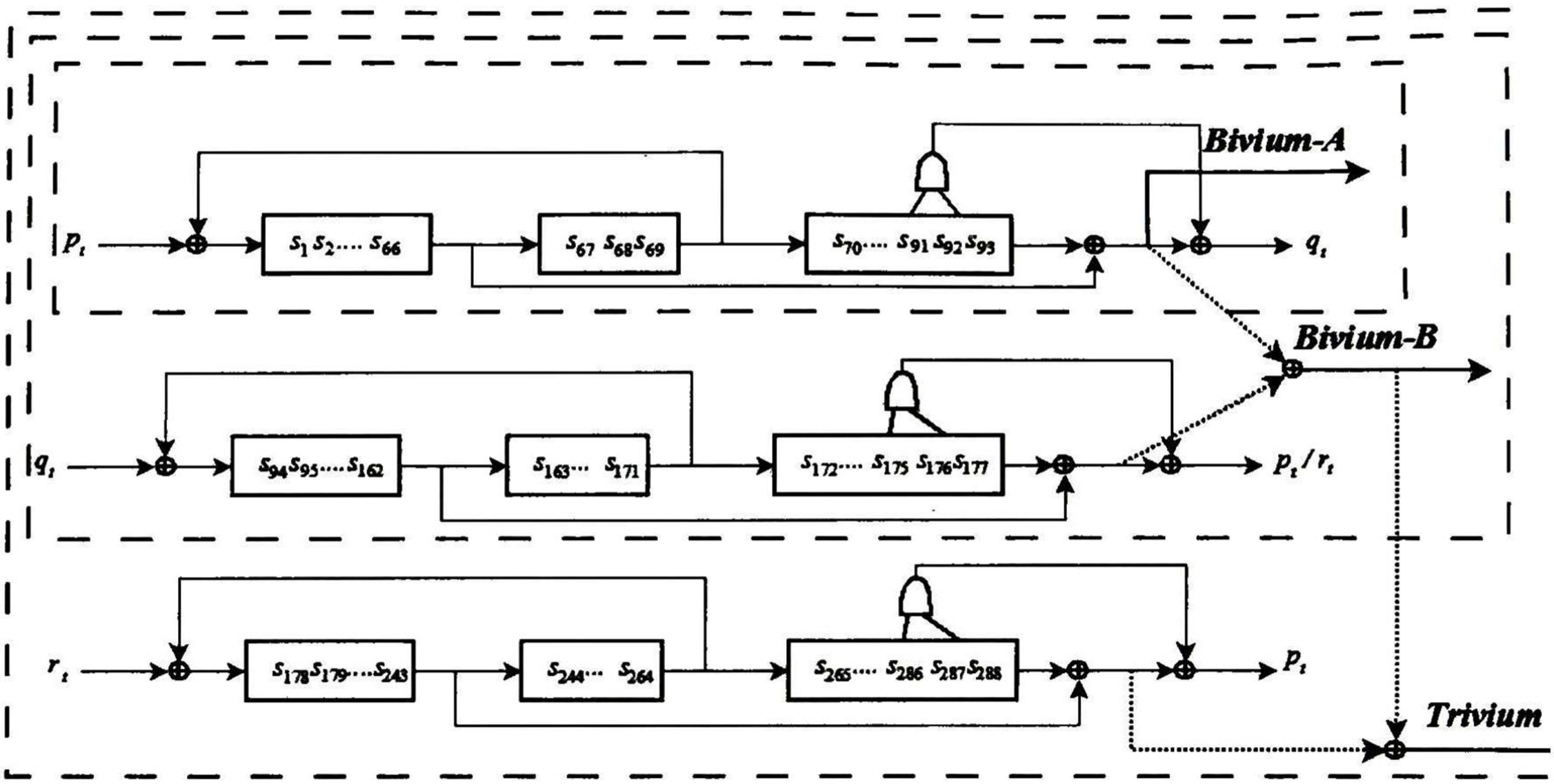


Figura 4.1: Esquema de Trivium, Bivium-A y Bivium-B

mensaje cifrado. Para desencriptar, el receptor produce el mismo keystream y hace la suma módulo 2 (XOR) bit a bit para obtener el mensaje; en otras palabras $c_i \oplus z_i = m_i$.

La Figura 4.1 muestra la estructura interna de Trivium, de la cual también obtenemos las variantes llamadas Bivium-A y Bivium-B.

Pseudocódigos de Trivium

La etapa de inicialización de Trivium consta de 4 rondas, una ronda consta en dejar correr Trivium 288 veces el siguiente estado, el Algoritmo 3 muestra la etapa de inicialización.

Una vez que se llevó a cabo la etapa de inicialización de Trivium se puede utilizar el keystream para sumar bit a bit el i-ésimo bit del mensaje con el i-ésimo bit del keystream; $m_i \oplus z_i = c_i$. El Algoritmo 4 muestra como obtener el keystream.

Algoritmo 3: Etapa de inicialización de Trivium

Sea $(K_1, K_2, \ldots, K_{80})$ el vector con la clave secreta, $(IV_1, IV_2, \ldots, IV_{80})$ el vector inicial, entonces:

1: $(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, K_2, \ldots, K_{80}, 0, 0, \ldots, 0)$
2: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, IV_2, \ldots, IV_{80}, 0, 0, \ldots, 0)$
3: $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)$
4:
5: **for** $i = 1$ **to** $4 * 288$ **do**
6: $t_1 \leftarrow s_{66} \oplus s_{93}$
7: $t_2 \leftarrow s_{162} \oplus s_{177}$
8: $t_3 \leftarrow s_{243} \oplus s_{288}$
9: $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
10: $t_2 \leftarrow t_2 \oplus s_{175} \quad s_{176} \oplus s_{264}$
11: $t_3 \leftarrow t_3 \oplus s_{286} \quad s_{287} \oplus s_{69}$
12: $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
13: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
14: $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
15: **end for**

Algoritmo 4: Pseudocódigo de Trivium

Sea n la longitud del mensaje, entonces tenemos:

1: **for** $i = 1$ **to** n **do**
2: $t_1 \leftarrow s_{66} \oplus s_{93}$
3: $t_2 \leftarrow s_{162} \oplus s_{177}$
4: $t_3 \leftarrow s_{243} \oplus s_{288}$
5: $z_i \leftarrow t_1 \oplus t_2 \oplus t_3$
6: $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
7: $t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}$
8: $t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}$
9: $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
10: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
11: $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
12: **end for**

4.3.2. Descripción de Bivium

Bivium es un cifrado de flujo con 177 bits de estado interno (como muestra la Figura 4.1, Bivium puede ser visto como 2 LFSRs, uno de 93 bits y otro de 84 bits) y un tamaño de clave de 80 bits. El estado interno de Bivium se inicializa con la clave secreta, el vector de inicialización (IV) y ceros. Después, se deja correr 4 * 177 veces y comienza a producir bits de salida.

Bivium-A

Bivium-A es el cifrado más simple de los tres, y por lo tanto tiene el sistema de ecuaciones más sencillo de resolver. Bivium-A sólo necesita un LSFR (como se muestra en la Figura 4.1). El Algoritmo 5 muestra la inicialización de Bivium-A, mientras que el Algoritmo 6 muestra cómo obtener su keystream.

Algoritmo 5: Etapa de inicialización de Bivium

Sea $(K_1, K_2, \ldots, K_{80})$ el vector con la clave secreta, $(IV_1, IV_2, \ldots, IV_{80})$ el vector inicial, entonces:

1: $(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, K_2, \ldots, K_{80}, 0, 0, \ldots, 0)$
2: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, IV_2, \ldots, IV_{80}, 0, 0, \ldots, 0)$
3: $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)$
4:
5: **for** $i = 1$ **to** $4 * 177$ **do**
6: $\quad t_1 \leftarrow s_{66} \oplus s_{93}$
7: $\quad t_2 \leftarrow s_{162} \oplus s_{177}$
8: $\quad t_1 \leftarrow t_1 \oplus s_{91} \quad s_{92} \oplus s_{171}$
9: $\quad t_2 \leftarrow t_2 \oplus s_{175} \quad s_{176} \oplus s_{69}$
10: $\quad (s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots, s_{92})$
11: $\quad (s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
12: **end for**

Algoritmo 6: Pseudocódigo de Bivium-A

Sea n la longitud del mensaje, entonces tenemos:

1: **for** $i = 1$ **to** n **do**
2: $t_1 \leftarrow s_{66} \oplus s_{93}$
3: $t_2 \leftarrow s_{162} \oplus s_{177}$
4: $z_i \leftarrow t_2$
5: $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
6: $t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{69}$
7: $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots, s_{92})$
8: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
9: **end for**

Bivium-B

La única diferencia entre Bivium-A y Bivium-B es que el bit de salida suma la variable temporal t_1 y la variable temporal t_2, a diferencia de Bivium-A que sólo suma la variable temporal t_2.

Bivium B utiliza 2 LFSRs y el proceso de inicialización se toma en $4 * 177$ veces, por lo que tiene el mismo proceso de inicialización que Bivium-A (Algoritmo 5). Ya que se realizó la inicialización se comienza a obtener el keystream el cual sumar bit a bit módulo 2 con el mensaje. El Algoritmo 7 muestra el pseudocódigo de Bivium-B.

Algoritmo 7: Pseudocódigo de Bivium-B

Sea n la longitud del mensaje, entonces tenemos:

1: **for** $i = 1$ **to** n **do**
2: $t_1 \leftarrow s_{66} \oplus s_{93}$
3: $t_2 \leftarrow s_{162} \oplus s_{177}$
4: $z_i \leftarrow t_1 \oplus t_2$
5: $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
6: $t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{69}$
7: $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots, s_{92})$
8: $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
9: **end for**

4.3.3. Criptoanálisis lógico-algebraico a Bivium-B

En esta subsección primero mostramos una instancia del criptoanálisis lógico-algebraico a Bivium-B. Después, mostramos los resultados obtenidos de la aplicación de este criptoanálisis.

Instancia del criptoanálisis lógico-algebraico a Bivium-B

En nuestro criptoanálisis no consideramos la etapa de inicialización. El criptoanálisis a Bivium-B consta de cinco pasos, que describimos a continuación:

Paso 1. Obtener las primeras 176 ecuaciones de Bivium-B, como se vio en la Subsección 4.3.2. Este paso es el equivalente al *paso* 1 del criptoanálisis lógico-algebraico.

- **Paso 2.** Multiplicar las 176 ecuaciones por cada una de las 176 variables. Con esto tendremos 30976 ecuaciones nuevas. Este paso es el equivalente al *paso* 2 del criptoanálisis lógico-algebraico, el cual nos dice que tenemos que aplicar el paso 1 del algoritmo XL con $D = 3$.

Paso 3. Convertir las ecuaciones a su forma normal conjuntiva, como se explicó en la Sección 2.9.

Paso 4. Aplicar la técnica RECNF (Subsección 3.1).

Paso 5. Utilizar el SAT-Solver para encontrar un modelo que satisfaga a la forma normal conjuntiva agregando las restricciones del paso 4.

- Si la solución obtenida es una solución espuria, entonces buscar otro modelo.

- Si una vez obtenida una solución, alguna de las variables se encuentra indeterminada, es decir, no cuenta con algún valor asociado, entonces generar un nuevo sistema de ecuaciones partiendo de la solución obtenida y regresar al paso 3.

La Figura 4.2 muestra el diagrama de flujo de los pasos descritos con anterioridad.

Resultados del criptoanálisis lógico-algebraico hacia Bivium-B

Debido a que no conocemos con exactitud el comportamiento de este criptoanálisis ejecutamos 1,000 veces el criptoanálisis a Bivium-B asumiendo que conocemos el valor de diferentes variables. Ya que asumimos que conocíamos varias variables, al tiempo que tarda nuestro criptoanálisis debemos añadir el tiempo que tomaría el encontrar esas variables asumidas. Logramos romper Bivium-B sin conocer ninguna variable en 2 ocasiones en aproximadamente 6 horas y en otras dos ocasiones tuvimos que detener el criptoanálisis después de 2 días de no tener respuesta.

Para dar a conocer el valor de una variable al criptoanálisis lo que hacemos es lo siguiente:

- Si la variable V_i es 1, entonces agregar la cláusula (V_i).
- Si no, entonces agregar la cláusula $(\overline{V_i})$.

Realizamos el criptoanálisis 1,000 veces por cada número de variables conocidas, conociendo 50 variables, 45 variables, 40 variables, 35 variables y 30 variables. Las Figuras 4.3 a 4.6 muestran los detalles de los criptoanálisis conociendo 45, 40, 35 y 30 variables, respectivamente. La Tabla 4.1 muestra una comparación entre los tiempos obtenidos conociendo

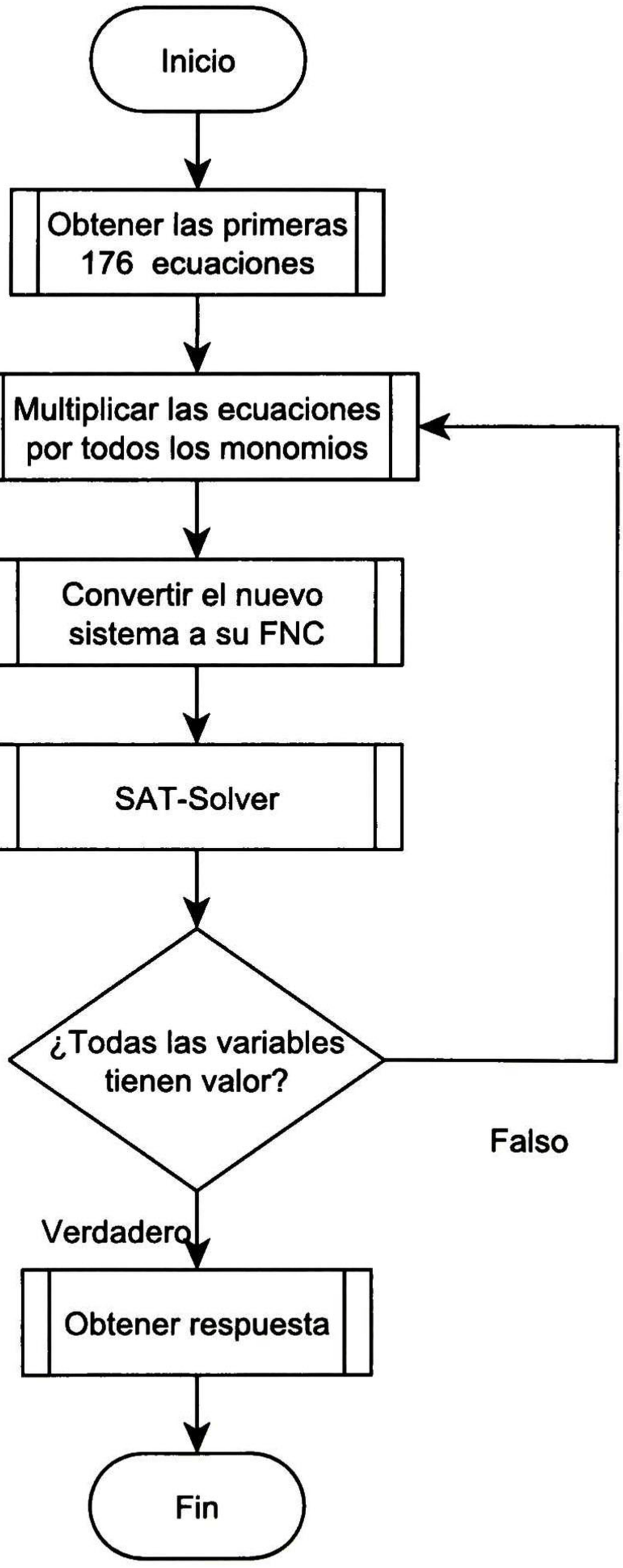


Figura 4.2: Diagrama de flujo del criptoanálisis lógico-algebraico aplicado a Bivium-B

algunas variables. El criptoanálisis conociendo 50 variables no varía mucho con respecto al criptoanálisis conociendo 45 variables.

En [Raddum, 2006], Raddum dice que aproximadamente $2^{13.3}$ claves se pueden buscar exhaustivamente. Por lo tanto, el mejor tiempo obtenido es adivinando 30 variables, así, tenemos un tiempo esperado de ejecución del criptoanálisis lógico-algebraico a Bivium-B aproximadamente de $2^{33.8773}$ segundos.

La Tabla 4.2 muestra una comparativa aproximada entre los criptoanálisis realizados a Bivium-B por Raddum [Raddum, 2006], McDonald et al. [McDonald et al., 2007], Simmonetti et al. [Simonetti et al., 2008] y Eibach et al. [Eibach et al., 2008].

Variables conocidas	Tiempo promedio	Tiempo menor	Tiempo mayor	Tiempo promedio conversión a FNC
45	1,610 ms	1,032 ms	5,854 ms	1,082 ms
40	4,358.4 ms	1,129 ms	14,756 ms	1,091 ms
35	6,370.1 ms	1,248 ms	19,693 ms	1,066 ms
30	41,741.4 ms	2,353 ms	123,800 ms	1,091 ms

Tabla 4.1: Resumen de los resultados obtenidos del criptoanálisis a Bivium-B

Autor	Tiempo (segundos)
Raddum[1]	2^{56}
McDonald et al.[1]	$2^{42.7}$
Simmonetti et al.[1]	2^{52}
Eibach et al.[1]	$2^{58.55}$
Lógico-algebraico	$2^{33.88}$

Tabla 4.2: Comparativa de los diferentes tiempos esperados del criptoanálisis a Bivium-B.

Las gráficas 4.7 a 4.10 muestran la comparación entre tiempos promedios, tiempos

[1] El autor no reporta sobre el(los) equipo(s) utilizado(s) durante el criptoanálisis.

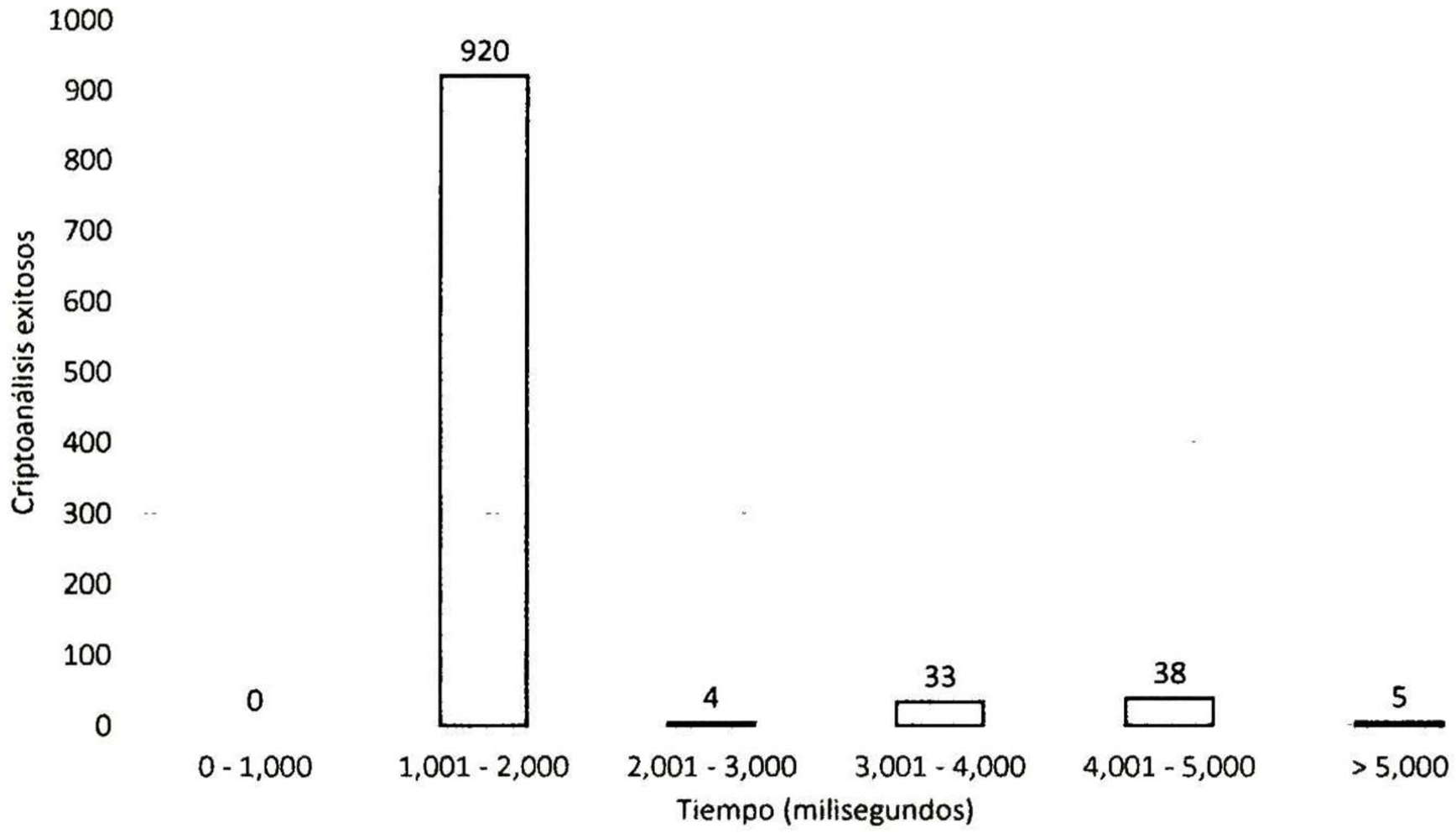


Figura 4.3: Criptoanálisis lógico-algebraico a Bivium-B conociendo 45 variables

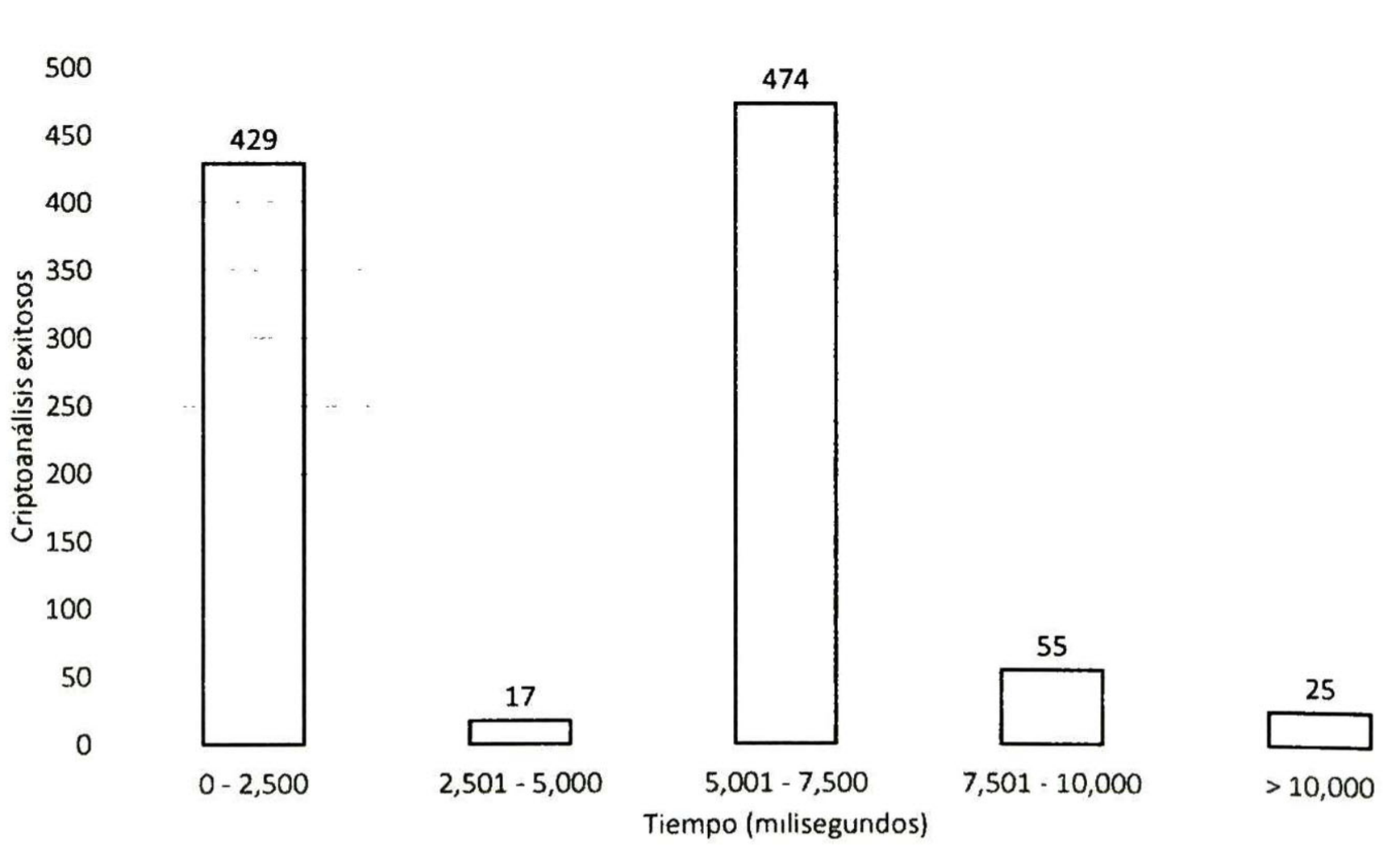


Figura 4.4: Criptoanálisis lógico-algebraico a Bivium-B conociendo 40 variables

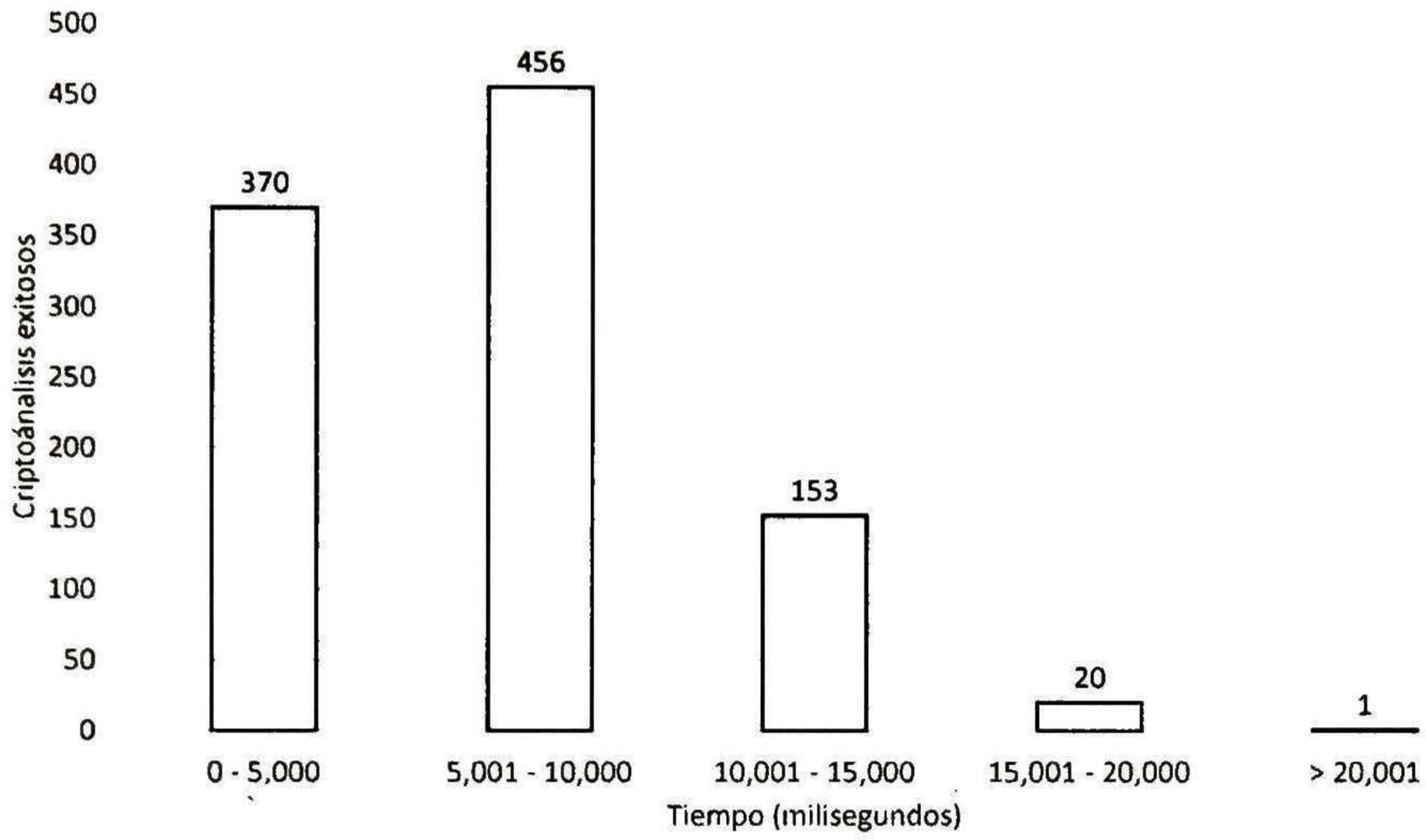


Figura 4.5: Criptoanálisis lógico-algebraico a Bivium-B conociendo 35 variables

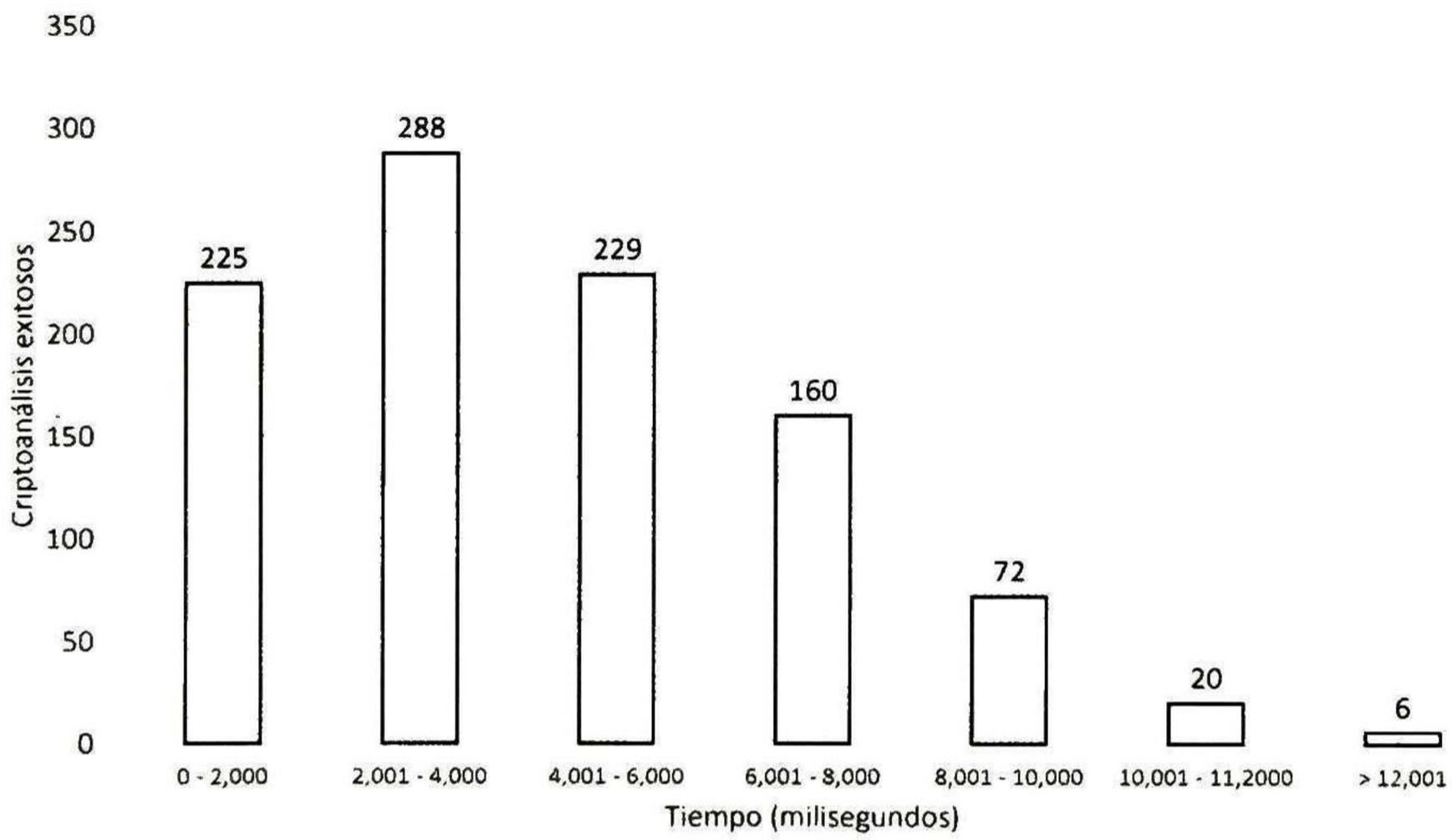


Figura 4.6: Criptoanálisis lógico-algebraico a Bivium-B conociendo 30 variables

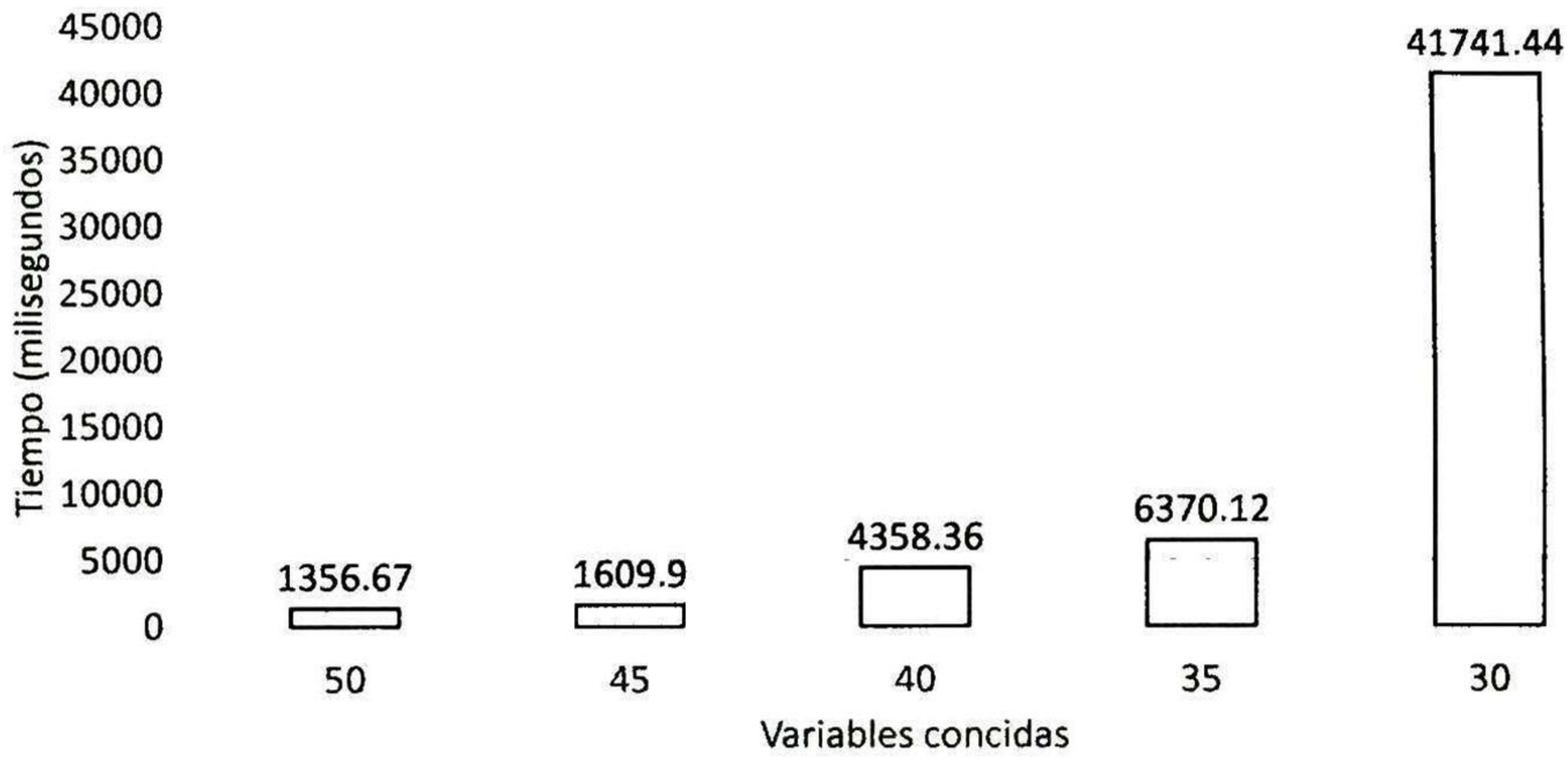


Figura 4.7: Comparación de tiempos promedios del criptoanálisis realizado a Bivium-B

menores, tiempos mayores y tiempos promedios de conversión en su forma normal conjuntiva, respectivamente.

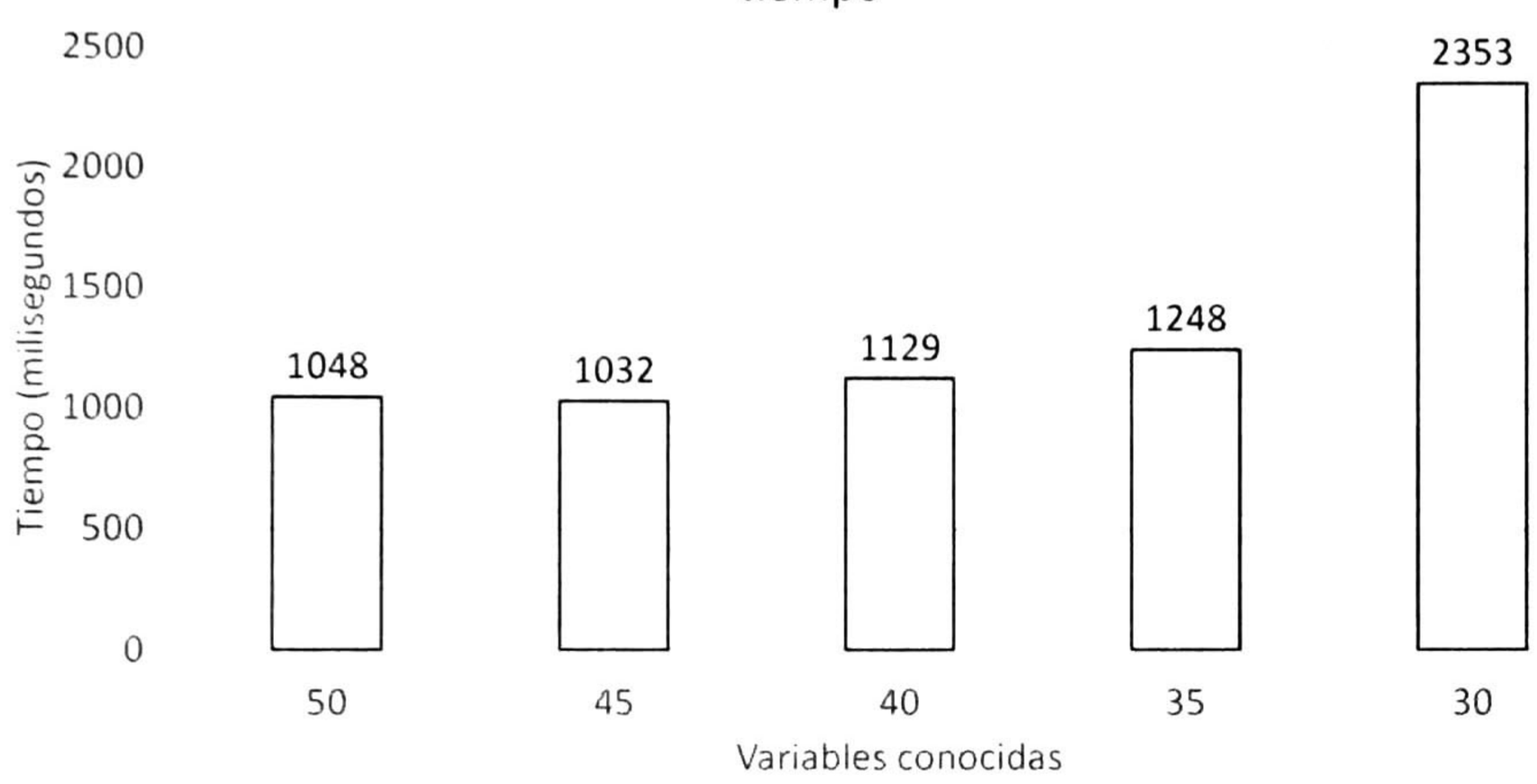


Figura 4.8: Comparación de tiempos menores encontrados del criptoanálisis realizado a Bivium-B

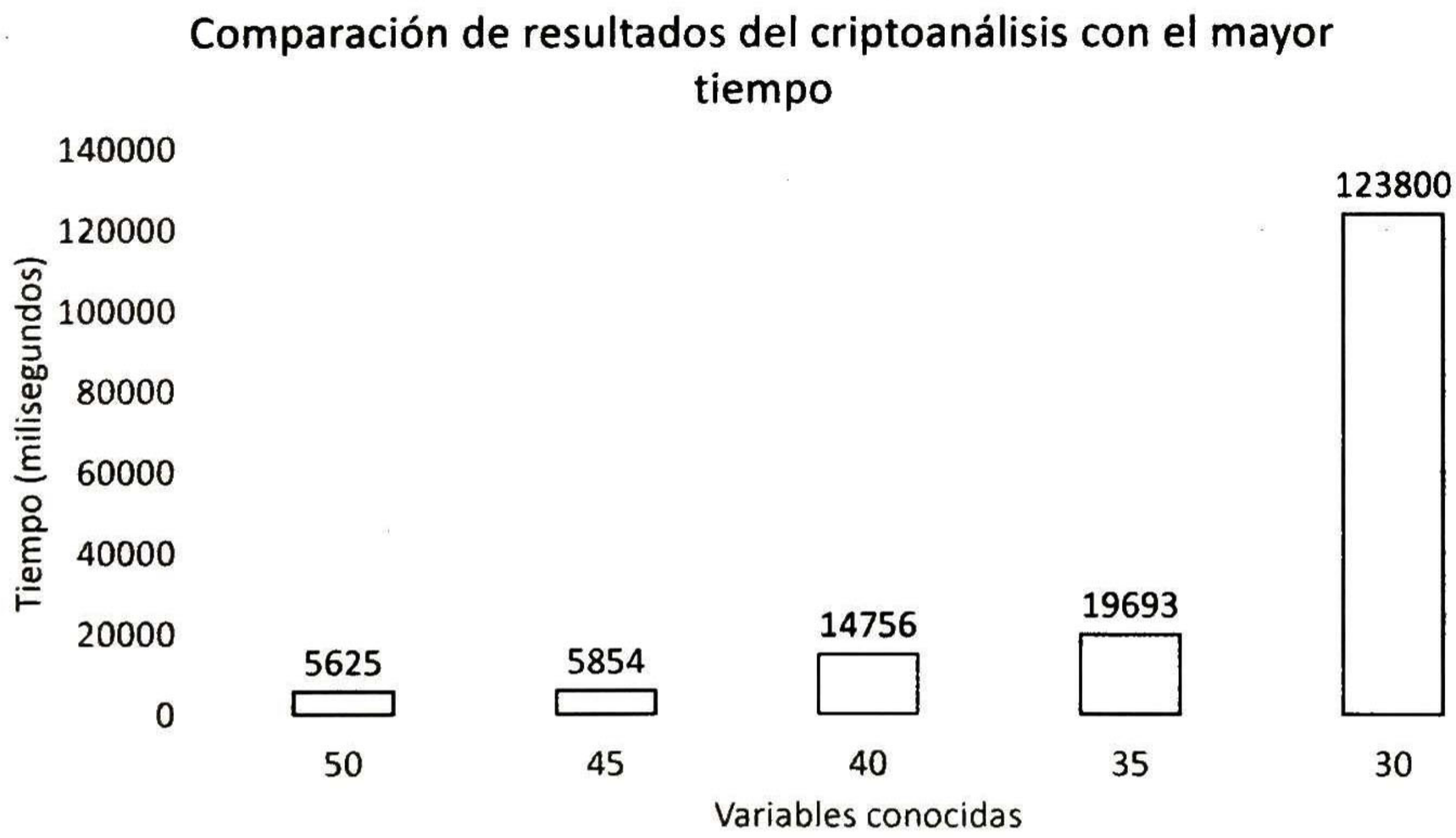


Figura 4.9: Comparación de tiempos mayores encontrados del criptoanálisis realizado a Bivium-B

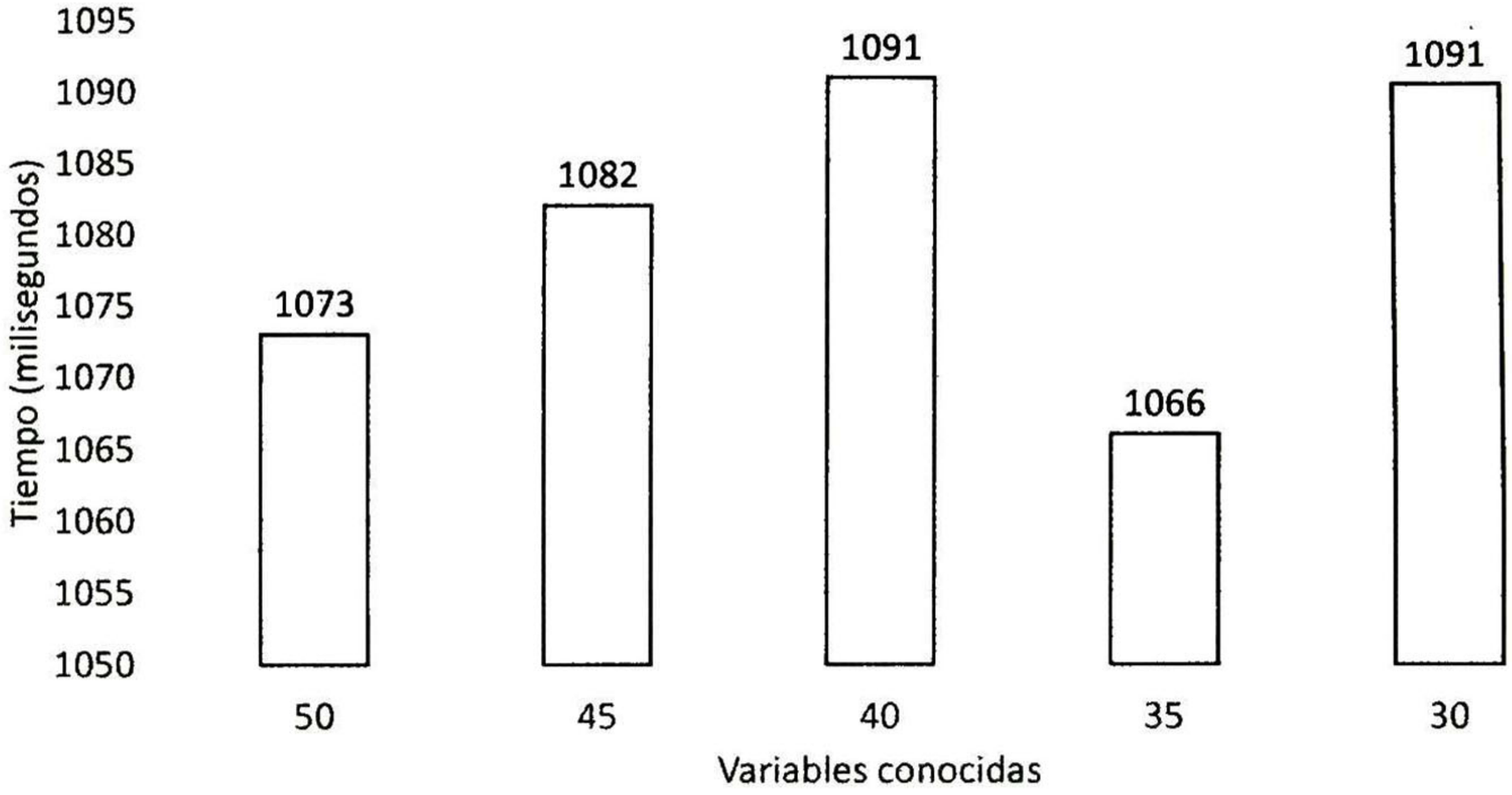


Figura 4.10: Comparación de tiempos promedios de conversión a FNC del criptoanálisis realizado a Bivium-B

4.4. FAPKC3

FAPKC es un *criptosistema de clave pública basado en autómatas finitos* (del acrónimo en inglés de *Finite Automata Public Key Cryptosystem*). FAPKC puede tanto encriptar como firmar digitalmente. En este trabajo nos enfocaremos al encriptado. Se encuentra basado en los autómatas finitos débilmente invertibles (*WIFAs*).

Bao e Igarashi explican en [Bao, 1995] las diferentes versiones de FAPKC llamadas FAPKC0, FAPKC1 y FAPKC2. En 1997, una nueva versión fue introducida en [Tao et al., 1997], llamada FAPKC3. En todos estos sistemas, la clave privada consiste de dos autómatas finitos cuyas inversas se pueden construir fácilmente y la clave pública es el autómata compuesto por los autómatas de la clave privada.

Se cree que es difícil obtener la descomposición de la clave pública para obtener los dos autómatas de la clave privada y que también es difícil calcular el autómata inverso

del autómata compuesto sin saber la descomposición; así cualquier usuario puede encriptar mensajes, pero no puede desencriptar el mensaje sin conocer la clave privada.

4.4.1. Algoritmo básico de FAPKC3

FAPKC3, como su nombre lo dice, es un criptosistema de clave pública. La clave privada del FAPKC3 consiste en dos autómatas finitos con memoria junto con dos estados iniciales, la clave pública es un autómata compuesto de los autómatas inversos de la clave privada y un estado inicial. En FAPKC3, tanto el texto plano como el texto cifrado son secuencias de l-tuplas de símbolos, donde $l \in \mathbb{N}$ fijo, tomadas de un conjunto de cardinalidad q. En este trabajo tomaremos a $q = 2$.

El texto cifrado es un poco mayor que el texto plano (por el retardo). El texto cifrado es la salida del autómata de la clave pública, este autómata tiene como entrada el texto plano. para la desencriptación, el receptor primero calcula una secuencia intermedia por un autómata finito (de la clave privada), posteriormente quita cierto retardo para la salida de este autómata, por último introduce la salida sin el retardo al segundo autómata para obtener el texto plano. Para mayor información se recomienda [Tao et al., 1997].

La Figura 4.11 muestra el diagrama básico de FAPKC3, donde $x_0, x_1, \ldots, x_n$ es el texto plano, $x_{n+1}, \ldots, x_l$ son los bits de retraso que se le agregan, $x'_0, x'_1, \ldots, x'_{l-t}, x'_{l-t+1}, \ldots, x'_l$ es el texto intermedio (entre el texto cifrado y el texto plano), $y_0, \ldots, y_l$ es el texto cifrado.

Para construir el criptosistema, primero seleccionamos q y l, después hacemos $l = m$. Σ, Γ son espacios vectoriales sobre $GF(2)$ de dimensiones l y m, respectivamente. Una vez seleccionados q y l construimos los autómatas $M_0, M_1, M_0^{-1}, M_1^{-1}$ como sigue:

1. Seleccionamos $h_0, k_0, \tau_0, \epsilon, h_1, \tau_1$ y después encontramos τ de la siguiente manera

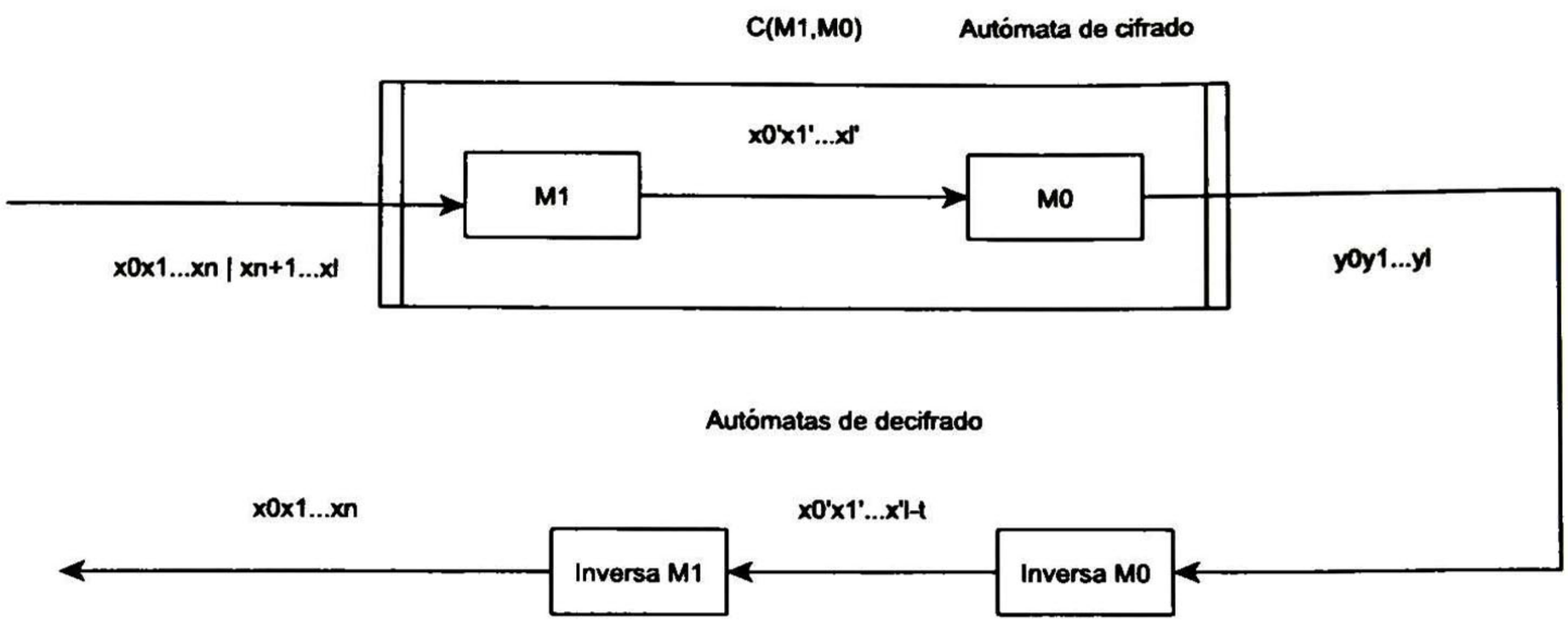


Figura 4.11: Diagrama básico de FAPKC3

$\tau = max(\tau_0, \tau_1, h_0)$.

2. Construimos el autómata finito $M_0 = (\Sigma, \Gamma, \Sigma^{k_0} \times \Gamma^{h_0}, \delta_0, \lambda_0)$, que es un autómata finito con orden de memoria-(k_0, h_0).

3. Construimos el autómata inverso de M_0, $M_0^{-1} = (\Gamma, \Sigma, \Gamma^{k_0+\tau_0} \times \Sigma^{h_0}, \delta_0^{-1}, \lambda_0^{-1})$, que es un autómata finito con orden de memoria-$(k_0 + \tau_0, h_0)$.

   La construcción de ambos autómatas debe satisfacer las siguientes condiciones:

   a) Para cualquier estado $s_0 = \langle y_{-1}, \ldots, y_{-k_0}, x_{-1}, \ldots, x_{-h_0} \rangle$ de M_0 y cualesquiera $x_0x_1, \ldots \in \Sigma$. Si $\lambda_0(s_0, x_0x_1 \ldots) = y_0y_1 \ldots$, entonces $\lambda_0^{-1}(s_0', y_{\tau_0}y_{\tau_0+1} \ldots) = x_0x_1 \ldots$, donde $s_0' = \langle x_{-1}, \ldots, x_{-h_0}, y_{-\tau_0-1}, \ldots, y_{-k_0} \rangle$. Es decir, para cualquier estado s en M_0 formado por $s = \langle y_{-1}, \ldots, y_{-k_0}, x_{-1}, \ldots, x_{-h_0} \rangle$, debe existir un s' en M_0^{-1} formado por $s' = \langle x_{-1}, \ldots, x_{-h_0}, y_{-\tau_0-1}, \ldots, y_{-k_0} \rangle$ talque (s, s') es un *par compatible con retraso* τ_0.

   b) Para cualquier estado $s' = \langle x_{-1}, \ldots, x_{-h_0}, y_{\tau_0-1}, \ldots, y_{-k_0} \rangle$ de M_0^{-1} y el estado $s = \langle y_{-\tau_0-1}, \ldots, y_{-\tau_0-k_0} \rangle$ de M_0, para cualesquiera $y_0, y_1, \ldots \in \Gamma$, si $\lambda_0^{-1}(s', y_0y_1 \ldots) = x_0'x_1' \ldots$, entonces $\lambda_0(s, x_0'x_1') = y_{-\tau_0}y_{-\tau_0+1} \cdots y_{-1}y_0y_1 \ldots$. Es decir, que M_0^{-1} es el

autómata inverso de M_0 y M_0 es el autómata inverso de M_0^{-1}.

4. Construimos un autómata finito con memoria de entrada-(h_1), $M_1 = (\Sigma, \Sigma, \Sigma^{h_1}, \delta_1, \lambda_1)$.

5. Construimos el autómata inverso de M_1, $M_1^{-1}(\Sigma, \Sigma, \Sigma^{h_1+\tau_1}, \delta_1^{-1}, \lambda_1^{-1})$, que es un autómata finito con orden de memoria-$(\tau_1 + h_1)$.

   La construcción de estos autómatas debe satisfacer las siguientes condiciones:

   a) Para cualquier estado s en M_1, $s = \langle x_{-1} \dots x_{-h_1} \rangle$ y cualesquiera $x_0 x_1 \dots \in \Sigma$, si $\lambda_1(s, x_0 x_1 \dots) = x'_0 x'_1 \dots$, entonces $\lambda_1^{-1}(s', x'_{\tau_1} x'_{\tau_1+1} \dots) = x_0 x_1 \dots$, donde $s' = \langle x_{-1}, \dots x_{-h_1}, x'_0, \dots, x'_{\tau_1-1} \rangle$. Es decir, M_1^{-1} es el autómata de inverso de M_1 y M_1 es el autómata inverso de M_1^{-1}

   b) Para cualquier estado $s' = \langle x_{-1}, \dots, x_{-h_1}, x'_{-1}, \dots, x'_{-\tau_1} \rangle$ de M_1^{-1} y para cualesquiera $x'_0, x'_1, \dots \in \Sigma$, si $\lambda_1^{-1}(s', x'_0 x'_1 \dots) = x_0 x_1 \dots$, entonces $\lambda_1(s, x_{\tau_1} x_{\tau_1+1} \dots) = x'_0 x'_1 \dots$, donde $s = \langle x_{\tau_1-1}, \dots, x_{\tau_1-h_1} \rangle$. Es decir, para cada estado s' en M_1^{-1} formado por $s' = \langle x_{-1}, \dots, x_{-h_1}, x'_{-1}, \dots, x'_{-\tau_1} \rangle$ su *par compatible* s este formado por $s = \langle x_{\tau_1-1}, \dots, x_{\tau_1-h_1} \rangle$.

6. Construimos el autómata compuesto $C(M_1, M_0) = (\Sigma, \Gamma, S, \delta, \lambda)$ a partir de M_0 y M_1.

7. Hacemos $\tau = max(\tau_0, \tau_1, h_0)$.

8. Obtenemos los estados de M_0^{-1} y M_1^{-1} como sigue:

   a) Seleccionamos aleatoriamente $y_{-1,s}, \dots, y_{-\tau-\tau_1,s} \in \Gamma$.

   b) Seleccionamos aleatoriamente un estado de M_0^{-1}, definido de la siguiente manera $s^0_{-\tau} = \langle x'_{-\tau-\tau_1-1,s}, \dots, x'_{-\tau-\tau_1-h_0,s}, y_{-\tau-\tau_1-1,s}, \dots, y_{-\tau-\tau_1-k_0-\tau_0,s} \rangle$.

   c) Seleccionamos aleatoriamente un estado de M_1^{-1}, definido de la siguiente manera $s^1_{-\tau} = \langle x_{-\tau-\tau_1-1,s}, \dots, x_{-\tau-\tau_1-h_0,s}, x''_{-\tau-\tau_1-1,s}, \dots, x''_{-\tau-\tau_1-k_0-\tau_0,s} \rangle$.

d) Calculamos:

$$\lambda_0^{-1}(s_\tau^0, y_{-\tau-\tau_1,s}, \ldots, y_{-1,s}) = x'_{-\tau-\tau_1,s}, \ldots, x'_{-1,s}$$
$$\delta_0^{-1}(s_{-\tau}^0, y_{-\tau-\tau_1,s}, \ldots, y_{-1,s}) = s_{0,s}$$
$$\lambda_1^{-1}(s_\tau^1, x'_{-\tau-\tau_1,s}, \ldots, x'_{-1,s}) = x_{-\tau-\tau_1,s}, \ldots, x_{-1,s}$$
$$\delta_1^{-1}(s_{-\tau}^0, x'_{-\tau-\tau_1,s}, \ldots, x'_{-1,s}) = s_{1,s}$$

e) Ponemos:

$$s_v^{out} = \langle y_{-1,s}, \ldots, y_{-k_0,s} \rangle$$
$$s_v^{in} = \langle x_{-1,s} \ldots, x_{-h_0,h_1+\tau_0+\tau_1,s} \rangle$$

f) Seleccionamos de manera aleatoria un estado de $C(M_1, M_0)$, definido de la siguiente manera

$s_e = \langle y_{-1,e}, \ldots, y_{-k,e}, x_{-1,e}, \ldots, x_{-h_0-h_1,e} \rangle$.

g) Calculamos:

$$\lambda_1(\langle x_{-h_0-1,e}, \ldots, x_{-h_0-h_1,e} \rangle, x_{-h_0-1,e}, \ldots, x_{-1,e}) = x'_{-h_0,e} \ldots x'_{-1,e}$$

h) Ponemos:

$$s_{1,d}^{out} = \langle x_{-1,e}, \ldots, x_{-h_1,e} \rangle$$
$$s_{0,d}^{out} = \langle x'_{-1,e}, \ldots, x'_{-h_0,e} \rangle$$

9. La clave pública es:

$C(M_1, M_0), s_v^{out}, s_v^{in}, s_e, \tau_0 + \tau_1$.

10. La clave privada es:

$M_0^{-1}, M_1^{-1}, s_{0,s}, s_{1,s}, s_{0,d}^{out}, s_{1,d}^{out}, \tau_0, \tau_1$

Encriptar

Sea $x_0, x_1, \ldots, x_n$ un texto plano que un usuario desea mandar de manera segura. Antes de hacerlo debe agregar $\tau_0 + \tau_1$ dígitos al texto plano, digamos que estos dígitos son $x_{n+1}, \ldots, x_{n+\tau_0+\tau_1}$. Después, utilizando la clave pública calcula el texto cifrado $y_0, y_1, \ldots, y_{n+\tau_0+\tau_1}$ como sigue:

$$\lambda(s_e, x_0 \ldots x_{n+\tau_0+\tau_1}) = y_0, y_1, \ldots, y_{n+\tau_0+\tau_1}$$

Desencriptar

Supongamos que recibimos el texto cifrado $y_0, y_1, \ldots, y_{n+\tau_0+\tau_1}$ y deseamos conocer el texto plano; para hacer esto debemos utilizar la clave secreta como sigue:

Primero utilizamos M_0^{-1} y $s_{0,d}^{out}$ y los dígitos $y_{-1,e}, \ldots, y_{-k_0,e}$ de s_e (la clave pública) y calculamos:

$$\lambda_0^{-1}(\langle x'_{-1,e}, \ldots, x'_{-h_0,e}, y_{\tau_0-1}, \ldots, y_0, y_{-1,e}, \ldots, y_{-k_0,e}\rangle, y_{\tau_0}, \ldots, y_{n+\tau_0+\tau_1}) = x'_0 \ldots x'_{n+\tau_1}$$

Después utilizamos M_1^{-1} y $s_{1,d}^{out}$ de la clave secreta y obtenemos el texto plano como sigue:

$$\lambda_1^{-1}(\langle x_{-1,e}, \ldots, x_{-h_1,e}, x'_{\tau_1-1}, \ldots, x'_0\rangle, x'_{\tau_1}, \ldots, x_{n+\tau_1}) = x_0 \ldots x_n$$

4.4.2. Descripción matricial de FAPKC3

Sea $t(x_0, \ldots, x_\epsilon)$ una función lineal de $\Sigma^{\epsilon+1}$ a Σ, donde ϵ es un entero pequeño positivo. Entonces podemos expresar los autómatas de la siguiente manera:

M_0:

$$y_i = \sum_{j=1}^{k_0} A_j y_{i-j} + \sum_{j=0}^{h_0} B_j x_{i-j}$$

M_0^{-1}:

$$x_i = \sum_{j=1}^{h_0} A'_j x_{i-j} + \sum_{j=0}^{\tau_0+k_0} B'_j y_{i-j}$$

M_1:

$$x'_i = \sum_{j=0}^{h_1} F_j x_{i-j} + \sum_{j=0}^{h_0-\epsilon} F'_j t(x_{i-j}, \ldots, x_{i-j-\epsilon})$$

M_1^{-1}:

$$x_i = \sum_{j=1}^{h_1} F''_j x_{i-j} + \sum_{j=1}^{\epsilon} F''_{h_1+j} t(x_{i-j}, \ldots, x_{i-j-\epsilon}) + \sum_{j=0}^{\tau_1} D_j x'_{i-j}$$

para $i = 0, 1, 2, 3, \ldots$ Donde $A_i, A'_i, B_i, B'_i, F_i, F'_i, F''_i, D_i$ son matrices de tamaño $l \times l$ y x_i, y_i, x''_i son vectores columna de tamaño l, para $i = 1, 2, \ldots$.

Entonces, el autómata compuesto $C(M_1, M_0)$ está definido por el siguiente sistema de ecuaciones vectoriales:

$$y_i = \sum_{j=1}^{k_0} A_j y_{i-j} + \sum_{j=0}^{h_0+h_1} C_j x_{i-j} + \sum_{j=1}^{h_0+h_1-\epsilon} C'_j t(x_{i-j}, \ldots, x_{i-j-\epsilon}),$$

para $i = 0, 1, \ldots$. Donde

$$C_j = \sum_{k+n=j} B_k F_n$$

$$C'_j = \sum_{k+n=j} B_k F'_n$$

4.4.3. Criptoanálisis lógico-algebraico a FAPKC3

Nuestro criptoanálisis está basado en la inversión por la descomposición de una matriz polinomial. La idea principal es obtener las matrices Bs, Fs y F's a partir de las matrices Cs. Bao e Igarashi [Bao, 1995] dieron la idea de este ataque por inversión cuando descompusieron la matriz polinomial para romper FAPKC0. El ataque realizado por Bao e Igarashi funciona siempre y cuando el retardo del autómata no lineal sea 0. Es por eso que en FAPKC3 el autómata no lineal tiene retraso.

Instancia del criptoanálisis lógico-algebraico a FAPKC3

Este criptoanálisis está basado en siete pasos:

Paso 1: Obtener ecuaciones cuadráticas simultaneas de la forma

$$C_j + \sum_{k+n=j} B_k F_n = 0$$

$$C'_j + \sum_{k+n=j} B_k F'_n = 0$$

- **Paso 2:** Aplicar el paso 1 del algoritmo XL con $D = 3$.

- **Paso 3:** Utilizando las definiciones de M_0 y M_1, generar un sistema de ecuaciones para encontrar $M'_0, M'^{-1}_0, M'_1, M'^{-1}_1$ (que no necesariamente son los autómatas iniciales) de la siguiente manera:

M_0:

$$y_i = \sum_{j=1}^{k_0} A_j y_{i-j} + \sum_{j=0}^{h_0} B_j x_{i-j}, i = 0, 1, \ldots, 2^{k_0+h_0}$$

$$x_i = \sum_{j=1}^{h_0} A'_j x_{i-j} + \sum_{j=0}^{\tau_0+k_0} B'_j y_{i-j}, i = 0, 1, \ldots, 2^{h_0+\tau_0+k_0}$$

M_1:

$$x'_i = \sum_{j=0}^{h_1} F_j x_{i-j} + \sum_{j=0}^{h_0-\epsilon} F'_j t(x_{i-j}, \ldots, x_{i-j-\epsilon}), i = 0, 1, \ldots, 2^{h_1}$$

$$x_i = \sum_{j=1}^{h_1} F''_j x_{i-j} + \sum_{j=1}^{\epsilon} F''_{h_1+j} t(x_{i-j}, \ldots, x_{i-j-\epsilon}) + \sum_{j=0}^{\tau_1} D_j x'_{i-j}, i = 0, 1, \ldots, 2^{h_1+\tau_1+\epsilon}$$

Por cada estado de M_0 generamos l ecuaciones y por cada estado de M_1 generamos l ecuaciones más. Así, necesitamos verificar si la máquina M'_0 y M'_1 obtenidas son WIFAs y en caso de que sean obtener $M_0^{-1\prime}$ y $M_1'^{-1}$, los cuales son inversas débil de M'_0 y M'_1, respectivamente.

- **Paso 4**: Convertir el sistema de ecuaciones a su FNC (Sección 2.9).

  Paso 5: Utilizar la técnica RECNF (Sección 3.1).

  Paso 6: Utilizar un SAT-Solver para obtener una respuesta.

- **Paso 7**: Si se tiene al menos una variable indeterminada, regresar al paso 2 con la solución que se tiene.

Obtener una respuesta: Después que aplicamos el paso 6 obtenemos una respuesta (del SAT-Solver) y esta respuesta es $M_0, M_0^{-1}, M_1, M_1^{-1}$ Así, hemos roto FAPKC3.

Para resumir lo dicho anteriormente, la Figura 4.12, muestra un diagrama de flujo del criptoanálisis lógico-algebraico a FAPKC3.

Resultados del criptoanálisis lógico-algebraico hacia FAPKC3

Como una instancia de FAPKC3 utilizamos un ejemplo propuesto por Meskanen [Meskanen et al., 2001], con $q = 2, l = 3, h_0 = 1, h_1 = 2, k_0 = 2, \tau_0 = 1, \epsilon = 1, \tau_1 = 2, \tau = 2$.

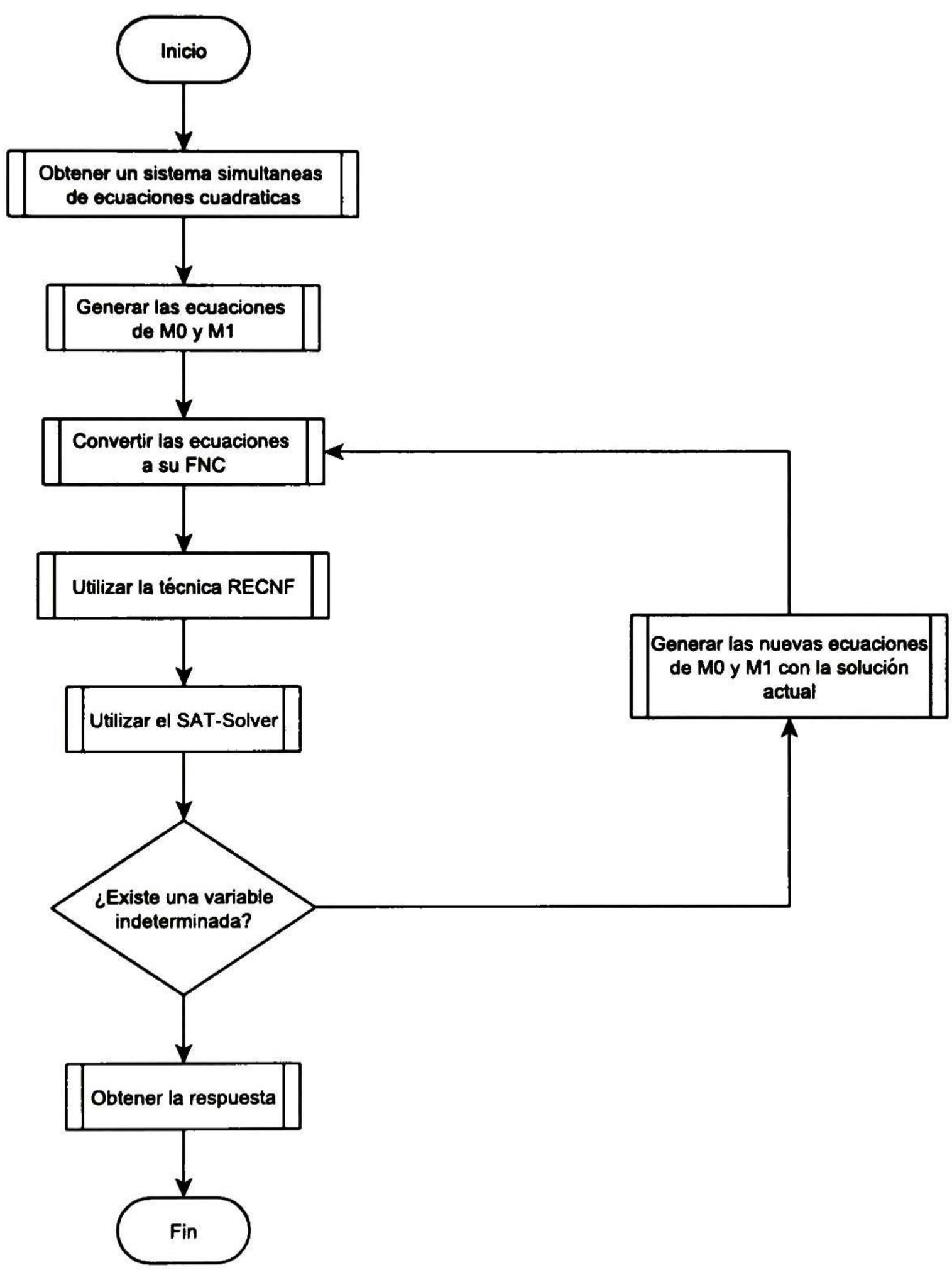


Figura 4.12: Diagrama de flujo del criptoanálisis lógico-algebraico a FAPKC3

Sea $S(x_i, x_{i+1})$ una función que multiplica componente a componente (módulo 2). Entonces, los autómatas son los siguientes:

M_0:

$$y_i = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} y_{i-1} + \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} y_{i-2} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_i + \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} x_{i-1}$$

M_0^{-1}:

$$x_i = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_{i-1} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} y_{i+1} + \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} y_i + \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} y_{i-1} + \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} y_{i-2}$$

M_1:

$$x'_i = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} x_i + \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} x_{i-1} + \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} x_{i-2} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} S(x_i, x_{i-1}) + \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} S(x_{i-1}, x_{i-2})$$

M_1^{-1}:

$$x_i = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} x_{i-1} + \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_{i-2} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} S(x_i, x_{i-1}) + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} x'_i +$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} x'_{i-1} + \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} x'_{i-2}$$

Con estos parámetros, si sólo hubiéramos querido encontrar M_0 y M_1 (utilizamos 63 variables) los encontramos rápidamente; pero cuando intentamos encontrar M_0^{-1} y M_1^{-1} tomó un tiempo considerable (después de dos días detuvimos el computo). Tao escribió en [Tao et al., 1997] lo siguiente:

Si se pueden descomponer las matrices B_j, F_j y F'_j de $C(M_1, M_0)$, entonces se puede obtener la descomposición de M_0 y M_1. Y fácilmente se puede encontrar M_0^{-1} y M_1^{-1} a partir de M_0 y M_1, respectivamente. Pero nunca menciona como lograr esto.

Entonces, si existe un procedimiento para encontrar las inversas de M_0 y M_1 a partir de M_0 y M_1, entonces podemos concluir que el sistema FAPKC3 es inseguro.

Meskanen [Meskanen et al., 2001] muestra, con un ataque, que el método propuesto por Tao [Tao, 2008] para generar autómatas debilmente invertibles es vulnerable; en el mismo trabajo, Meskanen, propone un nuevo método para generar autómatas debilmente invertibles con el cuál su ataque se vuelve ineficiente.

Con nuestro ataque no fue posible romper por completo el criptosistema FAPKC3, ya que si solamente necesitamos encontrar dos autómatas, M'_0 y M'_1, tales que la composición de estos autómatas es la misma que la composición con los autómatas originales, esto es, $C(M_1, M_0) = C(M'_1, M'_0)$, entonces el criptoanálisis se realiza rápidamente.

Capítulo 5

Conclusiones y trabajo futuro

Durante el transcurso de nuestro proyecto de tesis nos pusimos como objetivo diseñar una nueva técnica de criptoanálisis basado en dos criptoanálisis: (1) el criptoanálisis algebraico y (2) el criptoanálisis lógico. Logramos el objetivo propuesto con el diseño del criptoanálisis lógico-algebraico. Mostramos dos instancias del criptoanálisis lógico-algebraico, una hacia Bivium-B y otra hacia FAPKC3, logrando romper el criptosistema Bivium-B completamente y FAPKC3 en cierta parte, por lo que queda abierto a la investigación qué tan poderoso es este ataque.

Desarrollamos una nueva técnica (RECNF Sección 3.1), la cuál es una contribución propia; esta técnica nos ha dado resultados significativos al criptoanálisis lógico-algebraico.

La herramienta desarrollada se implementó en el lenguaje JAVA, que no es el más eficiente, es por eso que queda abierto a la investigación si se puede mejorar el tiempo si el criptoanálisis se implementa en el lenguaje C/C++. Los creadores de SAT4j [Le Berre et al., 2010], que es un SAT-Solver implementado en JAVA, mencionan que es hasta 4 veces más lento que MiniSAT 2.0, así que también queda abierto a la investigación si utilizando un SAT-Solver más eficiente (Como MiniSAT 2.0) el tiempo esperado de ejecución del criptoanálisis lógico-algebraico disminuye.

Las pruebas realizadas a Bivium fueron exitosas, ahora sólo queda implementar el criptoanálisis a Trivium y observar el tiempo esperado de ejecución para obtener la clave privada.

Gracias a la forma en la cual hemos propuesto el criptoanálisis lógico-algebraico tenemos las siguientes ventajas:

Agregamos restricciones, con las cuales eliminamos soluciones *espurias*.

Tenemos una nueva técnica de criptoanálisis a criptosistemas que se pueden modelar en un sistema de ecuaciones polinómicas multivariables.

Actualmente, existe una técnica, basada en la expansión de Shannon, para paralelizar la FNC en un SAT-Solver [Semenov et al., 2011], incluso existen SAT-Solvers que trabajan en paralelo, como el PMSAT [Gil et al., 2008]; es por eso que se deja como trabajo futuro el aplicar estas técnicas junto con el criptoanálisis lógico-algebraico y analizar las ventajas obtenidas, así, como observar si el tiempo esperado de ejecución disminuye.

Debido a que no fue posible romper el criptosistema FAPKC3 con este método, queda abierto a la investigación el proponer una nueva forma para modelar el criptosistema FAPKC3 y obtener M'_0, M'_1, M'^{-1}_0 y M'^{-1}_1 de una manera mas eficiente.

Muchos de los criptosistemas basan su seguridad en el problema de solucionar un sistema de ecuaciones (MQ problem, MP problem), es por eso que este tema a tomado mucha relevancia. El criptoanálisis lógico-algebraico ayuda al criptoanálisis de criptosistemas basados en el problema MP.

Bibliografía

[Bao, 1995] Bao, Feng, I. Y. (1995). Break finite automata public key cryptosystem. In *Automata, Languages and Programming*, pages 147–158. Springer.

[Bard, 2009] Bard, G. (2009). *Algebraic cryptanalysis*. Springer.

[Bard et al., 2007] Bard, G. V., Courtois, N. T., and Jefferson, C. (2007). Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF (2) via SAT-solvers.

[Biham y Shamir, 1991] Biham, E. and Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72.

[Courtois, 2003] Courtois, N. T. (2003). Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In *Information security and cryptology ICISC 2002*, pages 182–199. Springer.

[Courtois y Meier, 2003] Courtois, N. T. and Meier, W. (2003). Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology EUROCRYPT 2003*, pages 345–359. Springer.

[Creignou y Daude, 1999] Creignou, N. and Daude, H. (1999). Satisfiability threshold for random XOR-CNF formulas. *Discrete Applied Mathematics*, 96:41–53.

[De Cannière, 2006] De Cannière, C. (2006). Trivium: A stream cipher construction inspired by block cipher design principles. In *Information Security*, pages 171–186. Springer.

[Diffie y Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Acoustics Speech and Signal Processing Information Theory*, 22(6):644–654.

[Eén y Sörensson, 2004] Eén, N. and Sörensson, N. (2004). An extensible SAT-solver. In *Theory and applications of satisfiability testing*, pages 502–518. Springer.

[Eibach et al., 2008] Eibach, T., Pilz, E., and Völkel, G. (2008). Attacking Bivium using SAT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2008*, pages 63–76. Springer.

[Gil et al., 2008] Gil, L., Flores, P., and Silveira, L. M. (2008). PMSat: a parallel version of MiniSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:71–98.

[Kipnis y Shamir, 1999] Kipnis, A. and Shamir, A. (1999). Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Advances in cryptology CRYPTO99*, pages 19–30. Springer.

[Kohavi y Jha, 2010] Kohavi, Z. and Jha, N. K. (2010). *Switching and finite automata theory*. Cambridge University Press.

[Le Berre et al., 2010] Le Berre, D., Parrain, A., et al. (2010). The SAT4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64.

[Massacci y Marraro, 2000] Massacci, F. and Marraro, L. (2000). Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1-2):165–203.

[McDonald et al., 2007] McDonald, C., Charnes, C., and Pieprzyk, J. (2007). Attacking Bivium with miniSAT. *ECRYPT Stream Cipher Project*, 3.

[Meskanen et al., 2001] Meskanen, T. et al. (2001). On finite automaton public key cryptosystems. Technical Report 408, Turku Centre for Computer Science.

[Paar y Pelzl, 2009] Paar, C. and Pelzl, J. (2009). *Understanding cryptography: a textbook for students and practitioners.* Springer.

[Raddum, 2006] Raddum, H. (2006). Cryptanalytic results on Trivium. *eSTREAM, ECRYPT Stream Cipher Project, Report*, 39:2006.

[Rivest et al., 1978] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Sakalauskas, 2012] Sakalauskas, E. (2012). The multivariate quadratic power problem over Zn is NP-Complete. *Information Technology And Control*, 41(1):33–39.

[Semenov et al., 2011] Semenov, A., Zaikin, O., Bespalov, D., and Posypkin, M. (2011). Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system. In *Parallel Computing Technologies*, pages 473–483. Springer.

[Shannon, 1949] Shannon, C. E. (1949). Communication theory of secrecy systems* *Bell system technical journal*, 28(4):656–715.

[Simonetti et al., 2008] Simonetti, I., Faugere, J.-C., and Perret, L. (2008). Algebraic attack against Trivium. In *First International Conference on Symbolic Computation and Cryptography, SCC*, volume 8, pages 95–102.

[Smart, 2003] Smart, N. P. (2003). *Cryptography: an introduction.* McGraw-Hill New York.

[Tao, 2008] Tao, R. (2008). *Finite automata and application to cryptography.* Springer.

[Tao et al., 1997] Tao, R., Chen, S., and Chen, X. (1997). FAPKC3: a new finite automaton public key cryptosystem. *Journal of Computer Science and Technology*, 12(4):289–305.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Criptoanálisis Lógico-algebraico a Cifrados de Flujo Basados en Autómatas

del (la) C.

Luis Ricardo PEÑA LLAMAS

el día 16 de Enero de 2015.

Dr. Deni Librado Torres Román
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González
Pico
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Raúl Ernesto González Torres
Investigador CINVESTAV 2C
CINVESTAV Unidad Guadalajara

Dr. Andrés Méndez Vázquez
Investigador CINVESTAV Guadalajara
2C
CINVESTAV Guadalajara