Criptoanálisis algebraico a cifrados en bloques
Algebraic cryptanalyis to block ciphers

Irene Martínez Ferrer[1], Miguel Angel Borges Trenard[2], Mijail Borges Quintana[2*]

Resumen El criptoanálisis algebraico (**CA**) es una técnica de ataque a tomar en consideración sobre diversos esquemas de cifrado. El presente artículo trata sobre este tipo de ataque, cuando se lleva a cabo específicamente sobre los cifrados en bloques. Para ello, se aplica el Método de las Bases de Gröbner, como uno de las métodos utilizados en este criptoanálisis. Se profundiza en la metodología de trabajo del **CA** y también se diseñan herramientas computacionales que estarán disponibles para la continuación de esta investigación. Además, se ilustran las técnicas del **CA** con un ataque a un cifrado en bloques particular.

Abstract Algebraic cryptanalyis (**AC**) is an attack technique to be taken into consideration over diverse cipher schemes. The present paper deals with this type of attack, when it is carried out specifically over block ciphers. In order to do so, we apply the Gröbner Bases Method, as one of the methods that are used in this cryptanalysis. We get deep in the methodology of working of **AC** and also we design computational tools that will be available for the continuation of this research. Moreover, we illustrate the tecniques of **AC** with an attack to a particular block cipher.

Palabras Clave
criptoanálisis algebraico — base de Gröbner — cifrados en bloques

Keywords
algebraic cryptanalyis — Gröbner basis — block ciphers

[1] *Dirección de Criptografía, Ministerio del Interior, Santiago de Cuba, Cuba, ire.mtz@gmx.es.*
[2] *Departamento de Matemática, Universidad de Oriente, Santiago de Cuba, Cuba, borgestrenard2014@gmail.com, mijail@uo.edu.cu.*
***Autor para Correspondencia**

Introducción

Breaking a good cipher should require as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type[1].

Después de esas precursoras palabras, se conoce el criptoanálisis algebraico (**CA**) como el proceso de romper códigos mediante la resolución de sistemas de ecuaciones polinómicos. En sus inicios, el **CA** consistió en dos pasos principales: el primero es modelar el cifrado y alguna información adicional como un sistema de ecuaciones polinómicas (usualmente sobre $\mathbb{F}_2$), en el cual los bits de la clave sean incógnitas. El segundo paso, dada tal representación, es descubrir la clave secreta, lo cual debe ser equivalente a resolver el sistema de ecuaciones.

Existen diversos métodos para resolver sistemas de ecuaciones polinómicas, entre ellos podemos mencionar el Algoritmo XL, ElimLin, SAT-Solvers, el Método de las Bases de Gröbner (**BG**) y otros. La utilización de las **BG** en el **CA** no es un tema acabado, se tiene necesidad de encontrar las vías más eficaces para utilizar el ataque algebraico en diferentes direcciones, a saber, la modelación adecuada de los procesos de cifrado, descifrado, y generación de claves; la obtención de las soluciones de los sistemas polinómicos que se obtienen, y la combinación con otros tipos de ataques. El presente trabajo se basa en la tesis [7]. Se describe además en detalles cómo realizar un ataque algebraico a cifrados en bloques, resaltando dos momentos, la escritura de un sistema de ecuaciones que describa el proceso del cifrado en cuestión y la resolución de dicho sistema. Los experimentos se llevaron a cabo en una PC con las siguientes características:

Intel Core i7-4790, a 3.6 GHz, con 16 GB de RAM.

La estructura del trabajo es como sigue: La Sección 1 discute sobre uno de los algoritmos más eficientes para el cálculo de BG (F5). Posteriormente ilustra los pasos generales para realizar un **CA** (por vía directa o por encuentro en el medio). La Sección 2 se dedica a mostrar una combinación del ataque algebraico con la búsqueda exhaustiva. Luego, las conclusiones.

1. Criptoanálisis Algebraico

La idea general del ataque algebraico es conocida, pero la presentación de una metodología lo suficientemente detallada, fundamentada y práctica para llevar a cabo este tipo de ataque no es un tema sencillo ni resuelto completamente, la presentación que se hace en [11] se acerca bien al tratamiento de este tópico. El resumen que se presenta en esta sección utiliza esa referencia como punto de partida, incorporando enfoques y

[1] Claude E. Shannon. Communication Theory of Secrecy Systems. In Bell System Technical Journal 28, pages 656-715, 1949.

ejemplos desarrollados en esta investigación.

1.1 Algoritmo F5 en GAP

El Algoritmo F5 fue presentado por Faugère en 2002 (ver [6]). La versión original aparece en código de programación, por lo que resulta difícil entenderlo. En [13] se interpreta F5 como F5B (al estilo del Algoritmo de Buchberger), siendo esta variante más sencilla de entender e implementar.

Algorithm 1 (**Algoritmo F5 al estilo de Buchberger**).

Input: un conjunto de polinomos $\{f_1, f_2, ..., f_m\}$ de $K[X]$ y un orden admisible $\prec$.

Output: Una BG del ideal $\langle f_1, f_2, ..., f_m \rangle$ con respecto a $\prec$.

Begin
$F_i \leftarrow (e_i, f_i)$ para $i = 1, 2, ..., m$;
$B \leftarrow \{F_i \mid i = 1, 2, ..., m\}$;
$CP \leftarrow$ {par crítico $[F_i, F_j] \mid 1 \leq i < j \leq m$};
while CP sea no vacío **do**
  $cp \leftarrow$ seleccionar un par crítico de CP;
  $CP \leftarrow CP \setminus \{cp\}$;
  if cp no satisface el Criterio del Syzygy **ni** el Criterio de Reescritura **then**
    $SP \leftarrow$ el s-polinomio del par crítico de cp;
    $P \leftarrow$ el resultado de la F5-Reducción de SP por B, $SP \Rightarrow_B^* P$;
    if parte polinomial de P **no** es 0, $pol(P) \neq 0$, **then**
      $CP \leftarrow CP \cup$ {par crítico $[P, Q] \mid Q \in B$};
    end if
    $B \leftarrow B \cup P$;
  end if
end while
return {parte polinomial de $Q \mid Q \in B$};

En [9] se realizó un análisis comparativo en cuanto a la cantidad de s-polinomios calculados y el tiempo total de cómputo (para ideales asociados con códigos lineales e ideales sobre campos finitos), lo que permitió corroborar la eficiencia de este algoritmo comparado con el de Buchberger. Se implementó además una función que cuenta el número de formas canónicas que determina un ideal hasta una longitud dada, en caso que el ideal sea finito dimensional permitiría calcular todas las formas canónicas.

Esta implementación del Algoritmo F5, junto con otras técnicas de cálculo de bases de BG, pueden ser utilizadas en la experimentación con el criptoanálisis a cifrados en bloques, como en los casos que se muestran en las secciones siguientes.

1.2 Cifrado en Bloques de Juguete (Toy Block Cipher)

Presentaremos un sencillo cifrado en bloques iterativo, del tipo SPN, con texto claro y texto cifrado de 16 bits y dos rondas. El mismo consiste de una capa de substitución (S-caja), seguida por una capa de permutación y la adición con la clave. En el Algoritmo 2 se describe su funcionamiento. Cabe destacar que no se considera un algoritmo generador de claves de ronda, es decir, se considera la misma clave para cada ronda. Por comodidad, denotaremos a este cifrado por **TBC**.

En este algoritmo, SBox hereda la idea fundamental de la S-caja del AES (ver [12]). El procedimiento comienza por dividir un vector de estado $w := [w_0, ..., w_{15}]$ en cuatro bloques de cuatro bits consecutivos (nibles). Entonces, cada nible es considerado como un elemento del campo $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/(x^4 + x + 1)$. La S-caja toma estos elementos y retorna sus inversos en $\mathbb{F}_{16}$ (para entradas distintas de cero) y $0 \in \mathbb{F}_{16}$ si el nible de entrada es $[0,0,0,0]$. El elemento obtenido se representa como un nible y se reconstruye un bloque de 16 bits.

Por su parte, la capa de permutación está representada por la acción de Perm sobre los vectores de estado. El valor del bit en la posición i, $0 \leq i \leq 15$, se mueve a la posición $Pos(i)$, donde

$$Pos(i) = \begin{cases} 4 \cdot i \ mod\ 15, & 0 \leq i \leq 14, \\ 15, & i = 15. \end{cases}$$

Así, $Perm(w) = [w_{Pos(1)}, ..., w_{Pos(15)}]$. En [11] se sugiere cómo demostrar que con esa permutación se logra óptima difusión, en el sentido de que todo bit del texto cifrado depende de todos los bits del texto claro y de la clave.

Algorithm 2 : Algoritmo del cifrado **TBC**.

Input: Un texto claro p de 16-bit y una clave k de 16-bit.

Output: Un texto cifrado c de 16-bit.

Begin
Realizar la adición con la clave inicial: $w := p \oplus k = AdKey(p,k)$.
for $i = 1, \ldots, 2$ **do**
  Realizar la substitución S-box: $w := SBox(w)$.
  Realizar la permutación Perm: $w := Perm(w)$.
  Adicionar la clave: $w := AddKey(w,k) = w \oplus k$.
end for
El texto cifrado es $c := w$.
return c

1.3 Ecuaciones del cifrado TBC

Las ecuaciones serán escritas componente a componente de los bloques de 16 bits, por tanto, son ecuaciones sobre $\mathbb{F}_2$. Denotemos por $p = [p_0, \ldots, p_{15}]$ y $c = [c_0, \ldots, c_{15}]$ al texto claro y texto cifrado respectivamente (los cuales aparecen como parámetros del sistema), $k = [k_0, ..., k_{15}]$ el bloque de la clave desconocida. Sean, por otra parte, $x_i = [x_{i,0}, ..., x_{i,15}]$, $i = 0, 1$, el resultado de realizar la suma con la clave, $y_i = [y_{i,0}, \ldots, y_{i,15}]$, $i = 1, 2$, las salidas de las S-cajas, y $z_i = [z_{i,0}, ..., z_{i,15}]$, $i = 1, 2$, los resultados de la capa de permutación.

Es ahora posible escribir el proceso de cifrado como el siguiente sistema:

$$\begin{cases} x_0 = p+k, \\ y_i = SBox(x_{i-1}), i = 1,2, \\ z_i = Perm(y_i), i = 1,2, \\ x_1 = z_1 + k, \\ c = z_2 + k. \end{cases}$$

El algoritmo consta de tres funciones, estas son: adición con la clave, substitución (la cual se descompone en la aplicación de cuatro S-cajas de 4-bits), y la permutación. La adición con la clave puede ser representada de modo trivial y puede ser escrita a nivel de bit, por ejemplo, en la adición con la clave inicial: $x_{0,j} = p_j + k_j, 0 \leq j \leq 15$.

Con respecto a *Perm* se tiene que los bloques $z_i = Perm(y_i),\ i = 1,2$, pueden ser escritos como

$$z_{i,j} = y_{i,Pos(j)}, 0 \leq j \leq 15.$$

Una pregunta interesante es ¿cómo escribir ecuaciones sobre $\mathbb{F}_2$ que describan a la S-caja?. La clave para la respuesta está en que es posible concentrarse en escribir las ecuaciones sólo para una S-caja (ya que *SBox* está compuesta de cuatro S-cajas iguales y paralelas que realizan operaciones sobre $\mathbb{F}_{16}$).

Sean $a = [a_0, a_1, a_2, a_3]$ los bits de entrada de la S-caja y $b = [b_0, b_1, b_2, b_3]$ los bits de salida. Considerando los dos casos ($a \neq 0$, $a \in \mathbb{F}_{16}$) y ($a = 0_{\mathbb{F}_{16}}$) se obtienen las ecuaciones explícitas:

$$\begin{aligned} b_0 &= a_0a_1a_2 + a_1a_2a_3 + a_0a_2 + a_1a_2 + a_0 + a_1 + a_2 + a_3, \quad (1) \\ b_1 &= a_0a_1a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_3, \\ b_2 &= a_0a_2a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_2 + a_3, \\ b_3 &= a_1a_2a_3 + a_0a_3 + a_1a_3 + a_2a_3 + a_1 + a_2 + a_3. \end{aligned}$$

Ataque al TBC por vía directa (hacia adelante)

Se ilustra, con el auxilio del MAPLE 18, el procedimiento para escribir un sistema de ecuaciones que modelen el proceso de cifrado del **TBC** en 2 rondas.

G16 := GF(2, 4, $alpha^4 + alpha + 1$):

e := G16:-ConvertIn(alpha):

p := [seq(p[i], i=1..16)]:

c := [seq(c[i], i=1..16)]:

k := [seq(k[i], i=1..16)]:

Ronda 1:

Para la primera ronda de cifrado debemos obtener las expresiones siguientes (ver Sección 1.3):

$$\begin{cases} x_0 = p+k, \\ y_1 = SBox(x_0), \\ z_1 = Perm(y_1), \\ x_1 = z_1 + k. \end{cases}$$

El primer paso es $x_0 = p + k$, esto se logra así:

x0:=[seq(p[i], i=1..16)] + [seq(k[i], i=1..16)];

x0 := $[k_1+p_1, k_2+p_2, k_3+p_3, k_4+p_4, k_5+p_5, k_6+p_6,$
$k_7+p_7, k_8+p_8, k_9+p_9, k_{10}+p_{10}, k_{11}+p_{11}, k_{12}+p_{12},$
$k_{13}+p_{13}, k_{14}+p_{14}, k_{15}+p_{15}, k_{16}+p_{16}]$

Se debe particionar el vector $x0$ de 16 componentes en 4 nibles que hemos llamado $n[1,i]$, $i = 1,\ldots,4$, para denotar el nible i de la ronda 1.

n[1,1]:=[k[1]+p[1], k[2]+p[2], k[3]+p[3], k[4]+p[4]]:

n[1,2]:=[k[5]+p[5], k[6]+p[6], k[7]+p[7], k[8]+p[8]]:

n[1,3]:=[k[9]+p[9], k[10]+p[10], k[11]+p[11], k[12]+p[12]]:

n[1,4]:=[k[13]+p[13], k[14]+p[14], k[15]+p[15],
k[16]+p[16]]:

En la Sección 1.3 se demuestra que la S-caja queda totalmente modelada por el sistema de ecuaciones explícitas (1) y las relaciones de definición del campo con respecto a los $a_i's$.

Utilizando el sistema (1) de la Sección 1.3, se puede expresar cada nible de salida de la S-caja en función de las componentes del nible de entrada, es decir, para un nible arbitrario $[a_1, a_2, a_3, a_4]$, el nible B de salida de la S-caja tiene la forma siguiente:

B:=[a[1]*a[2]*a[3] + a[2]*a[3]*a[4] + a[1]*a[3] + a[2]*a[3]+
a[1] + a[2] + a[3] + a[4], a[1]*a[2]*a[4] + a[1]*a[2] +
a[1]*a[3] + a[2]*a[3] + a[2]*a[4] + a[4], a[1]*a[3]*a[4] +
a[1]*a[2] + a[1]*a[3] + a[1]*a[4] + a[3] + a[4],
a[2]*a[3]*a[4] + a[1]*a[4] + a[2]*a[4] + a[3]*a[4] + a[2] +
a[3] + e[4]]:

Entonces, para el nible

$$n[1,1] := [k[1]+p[1], k[2]+p[2], k[3]+p[3], k[4]+p[4]],$$

el correspondiente nible de salida puede obtenerse sustituyendo en el sistema B el valor de las componentes de $n[1,1]$, es decir:

$a[1] = n[1,1][1] = k[1]+p[1], a[2] = n[1,1][2] = k[2]+p[2],$
$a[3] = n[1,1][3] = k[3]+p[3], a[4] = n[1,1][4] = k[4]+p[4].$

las componentes de B tomarán la forma siguiente:

b[1, 1]:=[(k[1]+p[1])*(k[2]+p[2])*(k[3]+p[3])+(k[2]+p[2])
(k[3]+p[3])(k[4]+p[4])+(k[1]+p[1])*(k[3]+p[3])+(k[2]+
p[2])*(k[3]+p[3])+k[1]+p[1]+k[2]+p[2]+k[3]+p[3]+k[4]+
p[4], (k[1]+p[1])*(k[2]+p[2])*(k[4]+p[4])+(k[1]+p[1])*
(k[2]+p[2])+(k[1]+p[1])*(k[3]+p[3])+(k[2]+p[2])*(k[3]+
p[3])+(k[2]+p[2])*(k[4]+p[4])+k[4]+p[4],
(k[1]+p[1])*(k[3]+p[3])*(k[4]+p[4])+(k[1]+p[1])*(k[2]+
p[2])+(k[1]+p[1])*(k[3]+p[3])+(k[1]+p[1])*(k[4]+p[4])+
k[3]+p[3]+k[4]+p[4],
(k[2]+p[2])*(k[3]+p[3])*(k[4]+p[4])+(k[1]+p[1])*(k[4]+
p[4])+(k[2]+p[2])*(k[4]+p[4])+(k[3]+p[3])*(k[4]+p[4])+
k[2]+p[2]+k[3]+p[3]+k[4]+p[4]]:

Del mismo modo se obtienen los nibles de salida para $n[1,2], n[1,3], n[1,4]$, por ejemplo, el nible $b[1,2]$ se obtendrá sustituyendo la variable a_i por la componente $n[1,2][i]$, para $i = 1 \ldots 4$. Por su extensión solo se mostró el nible $b[1,1]$.

Una vez obtenidos los 4 nibles de salida, deben concatenarse, conformando el vector de 16 componentes que será la entrada de la función de permutación $z_1 = Perm(y_1)$.

y1 := [op(b[1, 1]), op(b[1, 2]), op(b[1, 3]), op(b[1, 4])]:

z1:=Perm(y1):

Se concluye la primera ronda realizando XOR con la clave:

$x_1 = z_1 + k$: x1:=AddKey(z1, [seq(k[i], i=1..16)]):

Ronda 2:

Al culminar la Ronda 1, se tiene el vector x_1, que es la entrada de la S-caja en la Ronda 2.

El próximo paso es construir los vectores que responden a las operaciones de la segunda ronda de cifrado. La primera expresión que se debe obtener es la salida de $SBox(x_1)$, recordemos que funciona particionando en 4 nibles el vector x_1 y luego, utilizando el sistema B escrito en la Ronda 1, se obtendrán los 4 nibles de salida, posteriormente se calcula y_2 de 16 componentes, concatenando estos nibles.

$$\begin{cases} y_2 = SBox(x_1), \\ z_2 = Perm(y_2), \\ x_2 = z_2 + k, \\ c = x_2. \end{cases}$$

Particionando x_1 en nibles:

n[2, 1]:=[x1[1], x1[2], x1[3], x1[4]]:

n[2, 2]:=[x1[5], x1[6], x1[7], x1[8]]:

n[2, 3]:=[x1[9], x1[10], x1[11], x1[12]]:

n[2, 4]:=[x1[13], x1[14], x1[15], x1[16]]:

El próximo paso es obtener $y_2 = SBox(x_1)$, para esto, debe hallarse la salida de la S-caja para $n[2,1], n[2,2], n[2,3], n[2,4]$. Ésto se logra de manera similar a como se procedió en la Ronda 1, la salida para cada uno de estos nibles puede ser expresada en función de las componentes del nible de entrada, como describe el sistema (1). Sustituyendo en B:

$a[1] = n[2,1][1] = x1[1], a[2] = n[2,1][2] = x1[2],$

$a[3] = n[2,1][3] = x1[3], a[4] = n[2,1][4] = x1[4],$

se obtiene el nible $b[2,1]$ y luego a $y_2 = SBox(x_1)$.

y2:=[op(b[2, 1]), op(b[2, 2]), op(b[2, 3]), op(b[2, 4])]:

Corresponde ahora aplicar la permutación al vector y_2:

z2:=Perm(y2):

Finaliza la segunda ronda con $x_2 = z_2 + k$:

x2 := AddKey(z2, [seq(k[i], i=1..16)]):

Como sólo son 2 rondas, ya se tiene el texto cifrado representado en un sistema de ecuaciones que depende de las componentes del texto claro y la clave (por su extensión no lo escribimos aquí). Si el número de rondas fuese mayor, sería necesario declarar variables de estado por ronda, que incrementarían el número de incógnitas y por tanto harían más complejo el procedimiento anterior. El texto cifrado c es la salida del **TBC**: c:=x2:

Veamos un ejemplo que muestra cómo, dado un par de texto claro y su cifrado correspondiente, es posible recuperar la clave, intentando resolver el sistema anterior.

Ejemplo 1 Sean

$$p1 = [1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1],$$

$$c1 = [0,1,1,1,0,1,1,1,0,0,1,1,1,1,1,0].$$

Con el recurso del MAPLE y de la expresión del cifrado obtenida anteriormente, veremos cómo hallar la clave utilizada. A continuación se definen en MAPLE $p1$, $c1$ y r.

p1:=[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]:

r:=2:

c1:=[0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0]:

Substituyendo los valores del texto claro $p1$, se obtiene un sistema t sólo en las componentes de la clave, seguidamente se construye s, uniendo a t las ecuaciones del campo, para $k_i, i = 1, \ldots, 16$:

```
t := c:
for i from 1 to 16 do
    p[i] := p1[i]:
od:
s := []:
for i from 1 to 16 do
    s := [op(s), (t[i]-c1[i]) mod 2]:
od:
s := [op(s), seq( k[ i ]^2 + k[ i ], i = 1..(16))]:
```

Utilizando *Basis* del paquete *Groebner* del MAPLE, se calcula la BG de s, como ahora se muestra.

Los experimentos se llevaron a cabo en una PC con las siguientes características: **Intel Core i7-4790, a 3.6 GHz, con 16 GB de RAM**.

```
with(Groebner):
infolevel[GroebnerBasis] := 2:
t:=time(); base:=Basis(r, plex(seq(k[i], i=1..16)),
characteristic=2); t:=time()-t;
```

Con estas instrucciones, se obtiene el siguiente resultado:

base := [k[16], k[15]^2 + k[15], k[14] + k[15], k[13] + 1,

k[15] + k[12], 1 + k[15] + k[11], k[10] + k[15], k[9],

k[15] + k[8], k[7], k[6], k[5], k[4] + 1, k[15] + k[3],

k[15] + k[2] + 1, k[1] + 1]:

A partir de la BG antes calculada se halla la clave k, utilizada para convertir el texto claro p_1 en el texto cifrado c_1, en 2 rondas de cifrado del **TBC**. Para hallar las soluciones en k_i, $i = 1,\ldots,16$, es útil la función *msolve* interna del MAPLE.

msolve(op(base),2);

$\{k_1 = 1, k_2 = 0, k_3 = 1, k_4 = 1, k_5 = 0, k_6 = 0, k_7 = 0,$

$k_8 = 1, k_9 = 0, k_{10} = 1, k_{11} = 0, k_{12} = 1, k_{13} = 1, k_{14} = 1,$

$k_{15} = 1, k_{16} = 0\}$,

$\{k_1 = 1, k_2 = 1, k_3 = 0, k_4 = 1, k_5 = 0, k_6 = 0, k_7 = 0,$

$k_8 = 0, k_9 = 0, k_{10} = 0, k_{11} = 1, k_{12} = 0, k_{13} = 1, k_{14} = 0,$

$k_{15} = 0, k_{16} = 0\}$.

Se han obtenido 2 claves consistentes, con las cuales pudo haber sido cifrado p_1:

$k_1 = [1,0,1,1,0,0,0,1,0,1,0,1,1,1,1,0]$,

$k_2 = [1,1,0,1,0,0,0,0,0,0,1,0,1,0,0,0]$.

En este caso, la clave que previamente habia sido utilizada para cifrar fue k_1.

TBC fue implementado en MAPLE 18, para cualquier valor de r, el mismo requiere de las funciones *Sbox*, *Perm*, *AddKey*, éstas a su vez dependen de *SboxNible*, *Pos*, *bd* y *db*, todas escritas para este trabajo.

Con el auxilio de *ToyCipherBlockEncryption* se puede verificar si el texto claro p_1 puede ser cifrado utilizando las claves k_1 y k_2 y dar lugar en ambos casos a c. Para el correcto funcionamiento de *ToyCipherBlockEncryption* debe inicializarse la hoja de cálculo del MAPLE, luego de esto ya se estará en condiciones de cifrar p_1 como sigue:

```
with(ListTools):
with(GroupTheory):
k1:=[1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0]:
k2:=[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0]:
p1:=[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]:
r:=2:
c1:=ToyCipherBlockEncryption(p1, k1, [k1, k1], r);
c1 := [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
c2:=ToyCipherBlockEncryption(p1, k2, [k2, k2], r);
c2 := [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
is(c1=c2);
true
```

En efecto, el texto cifrado de p_1 para la clave k_1 (c_1), coincide con el texto cifrado de p_1 para la clave k_2 (c_2).

Se ha descrito con un ejemplo sencillo la metodología para construir un sistema de ecuaciones que representa el proceso de cifrado de **TBC**. Para este ejemplo, se han obtenido expresiones al final de las 2 rondas de cifrado, que dependen de las componentes del texto claro y la clave. Si $r > 2$, repitiendo el procedimiento anterior se trabajaría con polinomios de mayor extensión, por lo que es común definir variables de estado por rondas y construir sistemas de ecuaciones dependiendo de estas variables intermedias. Dado un par de texto claro, texto cifrado, fue posible recuperar la clave utilizando los recursos del sistema MAPLE, comprobando este resultado con el Algoritmo *ToyCipherBlockEncryption* (implementación en MAPLE 18 que se realizó en este trabajo al Algoritmo 2).

Ataque al TBC por vía del encuentro en el medio

Para el par de texto claro-texto cifrado de la sección anterior, trataremos de hallar la clave utilizada por vía del encuentro en el medio.

Recordemos cómo se aplica el encuentro en el medio: Teniendo en cuenta que $Perm^{-1} = Perm$ y la transformación S-box es también inversible (con $0 \mapsto 0$ y $SBox^{-1} = SBox$), es posible entonces realizar las transformaciones inversas, partiendo del texto cifrado hasta llegar al texto claro, o bien, a estados intermedios (que es lo que se busca en el ataque por encuentro en el medio).

El encuentro en el medio que mostraremos consiste en, conocidos p y c, utilizar **TBC** hasta obtener z_1 (al final de la Ronda 1), como describe la sucesión de operaciones siguiente:

$$x_0 = p + k \to y_1 = Sbox(x_0) \to z_1 = Perm(y_1). \qquad (2)$$

Por otra parte, partiendo de c, también obtener z_1, como se muestra a continuación:

$$z_2 = c + k \to y_2 = Perm(z_2) \to x_1 = SBox(y_2) \to \\ \to z_1 = x_1 + k. \qquad (3)$$

Igualando los vectores construidos en (2) y (3), se tiene un sistema de ecuaciones que sólo depende de la clave. Dicho sistema puede intentar resolverse (por ejemplo, utilizando bases de Gröbner), y obtener la clave utilizada.

Ejemplo 2

En la Sección 1 se construyó la expresión del texto cifrado en términos de las componentes de la clave, debemos realizar ese mismo procedimiento hasta obtener z_1. Utilizando Maple 18 con las entradas definidas en la Sección 1, se tiene la expresión siguiente para z_1.

Corresponde entonces realizar el paso (2), eso se logra con las instrucciones siguientes:

z2:=[seq(c[i], i=1..16)]+[seq(k[i], i=1..16)]:

y2:=Perm1(z2):

n[2, 1]:=[y2[1], y2[2], y2[3], y2[4]]:

n[2, 2]:=[y2[5], y2[6], y2[7], y2[8]]:

n[2, 3]:=[y2[9], y2[10], y2[11], y2[12]]:

n[2, 4]:=[y2[13], y2[14], y2[15], y2[16]]:

Utilizando el sistema (2.1) de la Sección 1.3, podemos hallar $x_1 = SBox(y_2)$, análogo al proceder de la Sección 1. Se sustituye $a_i = n[2,1][i], i = 1\ldots4$, para el nible $n[2,1]$ y del mismo modo se trabaja con el resto de los nibles.

O sea, para el nible $n[2,1] := [y2[1], y2[2], y2[3], y2[4]]$, el correspondiente nible de salida puede obtenerse sustituyendo en el sistema B como sigue:

$a[1] = n[2,1][1] = y2[1], a[2] = n[2,1][2] = y2[2],$

$a[3] = n[2,1][3] = y2[3], a[4] = n[2,1][4] = y2[4].$

De este modo, $b[2,1]$ (nible de salida para la entrada $n[2,1]$ a la S-caja), podemos escribirlo como se muestra a continuación, notemos que tiene una estructura simple similar a los obtenidos al final de la primera ronda del **TBC**, con la diferencia de que en este caso el punto de partida ha sido el texto cifrado.

```
b[2,1]:=[k[1]*(k[5]+1)*k[9]+(k[5]+1)*k[9]*(k[13]+1)+
k[1]*k[9]+(k[5]+1)*k[9]+k[1]+k[5]+k[9]+k[13],
k[1]*(k[5]+1)*(k[13]+1)+k[1]*(k[5]+1)+k[1]*k[9]+(k[5]+
1)*k[9]+(k[5]+1)*(k[13]+1)+k[13]+1, k[1]*k[9]*(k[13]+
1)+k[1]*(k[5]+1)+k[1]*k[9]+k[1]*(k[13]+1)+k[9]+
k[13]+1, (k[5]+1)*k[9]*(k[13]+1)+k[1]*(k[13]+1)+
(k[5]+1)*(k[13]+1)+k[9]*(k[13]+1)+k[5]+1+k[9]]:
```

De manera similar resultan $b[2,2], b[2,3]$ y $b[2,4]$. Concatenando estos nibles, se obtiene el vector x_1. Luego se hace la suma XOR de x_1 con la clave obteniendo así a z_1. Para diferenciarlo del z_1 construido por **TBC** lo hemos denotado z_{11}.

```
x1 := [op(b[2,1]), op(b[2,2]), op(b[2,3]), op(b[2,4])]:

z11 := x1 + [seq(k[i], i=1..16)] mod 2:
```

Luego se substituyen los valores reales del texto claro y el texto cifrado en z_1 y z_{11} (los mismos valores que los del ejemplo de la sección anterior), se igualan ambas expresiones y se obtiene un sistema de ecuaciones s cuyas incógnitas son las componentes de la clave.

```
for i from 1 to 16 do
  p[i]:=p1[i]:
  c[i]:=c1[i]:
od:

s:=[]:

for i from 1 to 16 do
  s:=[op(s), z1[i]-z11[i] mod 2]:
od:

s mod 2
```

Este sistema puede ser resuelto por el Método de las Bases de Gröbner, añadiendo a s las ecuaciones del campo para $k_i, i = 1,\ldots,16$. Utilizando *Basis* del paquete *Groebner* del Maple, se calcula la base de Gröbner de s, luego *msolve* halla las soluciones del sistema asociado a la base de Gröbner obtenida, como se muestra a continuación.

```
r:=[op(s), seq( k[ i ]^2 + k[ i ], i = 1..16)]:

with(Groebner):

infolevel[GroebnerBasis] := 2:

t := time(); base := Basis( r, plex(seq(k[i], i=1..16) ),

characteristic=2 ); t := time()-t;
```

Se obtiene:

base := $[k_{16}, k_{15}+1, k_{14}^2+k_{14}, k_{13}k_{14}+k_{13}, k_{13}^2+k_{13},$

$1+k_{13}+k_{14}+k_{12}, k_{13}+k_{14}+k_{11}, k_{14}+k_{10}, k_9, k_8+1, k_7,$

$k_{13}+k_{14}+k_6, 1+k_{13}+k_5, 1+k_{13}+k_{14}+k_4, k_3+1,$

$1+k_{14}+k_2, k_1+1]$:

Ahora busquemos las soluciones:

msolve(op(base),2);

$\{k_1=1, k_2=0, k_3=1, k_4=0, k_5=1, k_6=1, k_7=0,$

$k_8=1, k_9=0, k_{10}=1, k_{11}=1, k_{12}=0, k_{13}=0, k_{14}=1,$

$k_{15}=1, k_{16}=0\},$

$\{k_1=1, k_2=0, k_3=1, k_4=1, k_5=0, k_6=0, k_7=0,$

$k_8=1, k_9=0, k_{10}=1, k_{11}=0, k_{12}=1, k_{13}=1, k_{14}=1,$

$k_{15}=1, k_{16}=0\},$

$\{k_1=1, k_2=1, k_3=1, k_4=1, k_5=1, k_6=0, k_7=0,$

$k_8=1, k_9=0, k_{10}=0, k_{11}=0, k_{12}=1, k_{13}=0, k_{14}=0,$

$k_{15}=1, k_{16}=0\}.$

Pudo observarse que *Basis* demoró menos tiempo en calcular la base de Gröbner para el encuentro en el medio que por la vía directa utilizada en la Sección 1 (sistema que representa el proceso de cifrado completo en las 2 rondas de cifrado). Ésto parece deberse a que en el encuentro en el medio se reduce la complejidad de los polinomios que intervienen en la modelación. Los resultados comparativos entre ambos tipos de ataque se resumen en la tabla siguiente. Para el encuentro en el medio se obtuvieron las soluciones siguientes:

Tabla 1. Tiempo que demora Basis (sec)

Basis	**vía directa**	**encuentro en el medio**
total time:	13.297 sec	1.297 sec

$k_1 := [1,0,1,0,1,1,0,1,0,1,1,0,0,1,1,0],$

$k_2 = [1,0,1,1,0,0,0,1,0,1,0,1,1,1,1,0],$

$k_3 := [1,1,1,1,1,0,0,1,0,0,0,1,0,0,1,0].$

Veamos cuáles son consistentes, utilizando la implementación de **TBC**, *ToyCipherBlockEncryption* (ver Sección 1):

```
k1:=[1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0]:
c11:=ToyCipherBlockEncryption(p1, k1, k1, r);
c11 := [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0]
is( c11 = c1 );
false
k2:=[1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0]:
c2:=ToyCipherBlockEncryption(p1, k2, k2, r);
c2 := [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0]
is( c2 = c1 );
true
k3:=[1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]:
c3:=ToyCipherBlockEncryption(p1, k3, k3, r);
c3 := [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0]
is( c3 = c11 );
false
```

Los resultados anteriores muestran que en el conjunto de soluciones calculado por *msolve* se encuentra la clave que fue utilizada para cifrar p (solución k_2). Además, k_1 y k_3 ilustran que utilizando el método del encuentro en el medio puede obtenerse la expresión del cifrado en un mismo punto intermedio z_1, pero ésto no significa que es equivalente el sistema obtenido al realizar el procedimiento completo, ronda por ronda, como se hizo en la Sección 1. Lo que si se garantiza que la clave utilizada tiene que ser una de las soluciones del sistema. En el ejemplo por vía directa hacia adelante fueron consistentes las 2 claves obtenidas al resolver el sistema que se construyó, lo cual era de esperar puesto que en ese caso se modeló completamente el proceso de cifrado.

2. Ataque algebraico por enfoque híbrido a cifrados en bloques

Los ataques algebraicos conducen a sistemas de ecuaciones polinómicas con gran número de ecuaciones e incógnitas, lo cual limita su aplicabilidad. En [3] se presenta un enfoque interesante y novedoso, para la resolución de sistemas polinómicos sobre campos finitos, el cual se basa en un compromiso entre la búsqueda exhaustiva y las técnicas de bases de Gröbner. Los autores del mencionado artículo calculan explícitamente la complejidad de este enfoque y muestran algunas aplicaciones en las que este método permitió romper desafíos de seguridad, para parámetros concretos, de varios esquemas criptográficos que se consideraban seguros. Ver también [4] y [5].

En [10] se brinda una implementación práctica, del método antes mencionado, en el sistema de álgebra computacional MAPLE (versión 17), se muestra cómo puede ser utilizado este algoritmo en ataques algebraicos a cifrados en bloques y cómo obtener una modelación polinómica de un cifrado en bloques del tipo SPN, se ilustra además la aplicación del enfoque híbrido con un ataque al cifrado HTC.

A continuación se describe brevemente el enfoque híbrido.

Enfoque Híbrido

Dado un campo finito $\mathbb{F}$, en teoría es posible encontrar las soluciones racionales de un sistema polinómico, con coeficientes en $\mathbb{F}$, mediante búsqueda exhaustiva, lo cual requiere $O(q^n)$ operaciones, donde q es el cardinal de $\mathbb{F}$ y n es el número de variables. El enfoque híbrido combina la búsqueda exhaustiva con el cálculo de bases de Gröbner, no calcula la base de Gröbner del sistema completo (que pudiese ser en muchos casos demasiado costoso o imposible bajo las condiciones de desarrollo actuales), calcula q^k bases de Gröbner de subsistemas que son obtenidos al fijar k variables. Se intuye que la ganancia que se obtiene, trabajando en sistemas con menos variables, pudiese superar la pérdida debido al recorrido exhaustivo en las variables fijadas. El problema es escoger el mejor compromiso, es decir, seleccionar el valor de k que minimice la complejidad del enfoque híbrido.

En lo que sigue, si $F = \{f_1, \ldots, f_m\}$, denotaremos por $V_{\mathbb{F}}(F)$ a la variedad de F sobre $\mathbb{F}$, es decir,

$$\{(z_1, \ldots, z_n) \in \mathbb{F}^n \,|\, \forall f \in F, f(z_1, \ldots, z_n) = 0\}.$$

Algorithm 3 : Enfoque Híbrido.

Input: $\mathbb{F}$, campo finito, $F = \{f_1, \ldots, f_m\} \subset \mathbb{F}[x_1, \ldots, x_n]$, $k \in \mathbb{N}$.

Output: $V_{\mathbb{F}}(F)$.

1: $S := \emptyset$;
2: **for** $(v_1, \ldots, v_k) \in \mathbb{F}^k$ **do**
3: Encontrar el conjunto solución $S_{\text{aux}} \subset \mathbb{F}^{n-k}$ del sistema $f(v_1, \ldots, v_k, x_{k+1}, \ldots, x_n) = 0$, $f \in F$;
4: $S := S \cup \{(v_1, \ldots, v_k, z_{k+1}, \ldots, z_n) \,|\, (z_{k+1}, \ldots, z_n) \in S_{\text{aux}}\}$;
5: **end for**
6: **return** S;

Es conveniente aclarar que se fijan las k primeras variables sólo para hacer más simple la escritura del algoritmo, ya que el mismo se puede aplicar en esencia fijando cualquier subconjunto de k de las variables.

En [5] se continúa profundizando en el enfoque híbrido presentado en [3], dando a conocer fórmulas asintóticas del comportamiento de este método bajo ciertas condiciones. En [10] Se contruye en MAPLE 17 la función HybridSolving, para realizar un ataque algebraico por enfoque híbrido. La utilización de HybridSolving requiere de los paquetes ListTools y Groebner del MAPLE.

Algorithm 4 : HybridSolving.

Input: F (conjunto de polinomios), X (lista con las variables), O (ordenamiento de términos), k (cantidad de variables fijadas), x_1, x_2 (límites entre los cuales se recorrerán los bloques binarios de longitud k, $x_1, x_2 \in [0, 2^k - 1]$).
Output: Soluciones del sistema $F = 0$.

```
1:  HybridSolving := proc( F, X, O, k, x1, x2 )
2:  local n, i, b, Z, x, S: n := nops( X ): S := []:
3:  for i from x1 to x2 do
4:     b := db( i, k ):
5:     Z := convert( convert( remove( x → member( x, [ 0, 1
       ] ),
       modp( subs( seq( X[ i ] = b[ i ], i =1..k ) , F ), 2 ) ), set
       ), list ):
6:     Z := Basis( Z, O( seq( X[ i ] , i = k+1.. n ) ), characte-
       ristic = 2 ):
7:     if is( Z <> [1] ) then
8:        Z := [ msolve( op( Z ) , 2 ) ]:
9:        Z := [ seq( seq( X[ i ] = b[ i ], i =1..k ), op( Z[ i ] ) ,
          i = 1..nops( Z ) ) ]:
10:       S := [ op( S ), op( Z ) ]:
11:    end if
12: end for
13: return S:
14: end proc:
```

Modelación polinómica del cifrado HTC

Una modelación polinómica de un cifrado en bloques es un conjunto de ecuaciones polinómicas que representa las operaciones que se realizan en el cifrado, estas ecuaciones dependen de los textos claros y cifrados (considerados como parámetros) y de la clave (variable o incógnita). Las ecuaciones deben reflejar los diversos pasos del cifrado, los cuales son típicamente (en el caso de las SPN), la adición con la clave, los movimientos a través de las S-cajas y transposiciones, así como el algoritmo de generación de las claves. También es usual introducir variables que representan los resultados por rondas (variables de estado).

Una SPN se obtiene mediante una sucesión de operaciones que son utilizadas en varios cifrados en bloques. Dado el texto claro y la clave, aplican un número fijo de veces (llamadas rondas) varias operaciones que, en general, consisten de substituciones S de los bits (mediante las llamadas S-cajas), permutaciones P de sus posiciones, así como sumas con las denominadas claves de rondas (las cuales típicamente se obtienen de la clave que introduce el usuario mediante algún algoritmo generador de ellas). Todas las rondas son iguales, salvo quizás la primera o la última. El descifrado se realiza invirtiendo el proceso (S y P tienen que ser inversibles) y aplicando las claves de ronda en orden invertido.

En [10] se ilustra el método híbrido con la modelación que se le realizó al cifrado HTC, denominado posteriormente en referencia a su autor (Heys Toy Cipher). HTC es un cifrado en bloques de 16 bits, del tipo SPN, con longitud de clave también de 16 bits. Tiene 4 S-cajas, de longitud binaria 4 (todas iguales), por tanto, el bloque de entrada se divide en 4 nibles y cada nible se pasa por la S-caja de la posición correspondiente. HTC consiste de 4 rondas, las 3 primeras son iguales y consisten de una suma con la clave del texto de salida de la ronda anterior (en la 1era ronda es la suma de la clave con el texto claro), seguida de la aplicación de las S-cajas y de una transposición en las posiciones de los bits de salida de las S-cajas. Finalmente, se realiza una cuarta ronda que no lleva transposición pero si una suma final con la clave de la ronda 4. Típicamente se ha considerado que todas las claves de ronda son iguales a la clave que introduce inicialmente el usuario. Para más detalles, puede consultarse [8], no obstante En la Tabla 2 se presenta la S-caja que se utiliza (expresada en hexadecimal).

De lo anterior se puede obtener, que si $[a,b,c,d]$ representa un nible de entrada arbitrario a la S-caja, entonces, el nible de salida viene dado por los siguientes polinomios:

$$\begin{aligned} 1+a+abc+b+bcd+ab+bc+d,\\ 1+a+b+bd+acd+cd+ac,\\ 1+abc+bd+abd+cd+ab+ad+bc+ac+c+d,\\ a+bd+acd+ad+c. \end{aligned} \quad (4)$$

Tabla 3. Notaciones

$X := [x_1, \ldots, x_{16}]$	bloque binario de longitud 16
$P := [p_1, \ldots, p_{16}]$	texto claro
$K := [k_1, \ldots, k_{16}]$	clave
$C := [c_1, \ldots, c_{16}]$	cifrado de P con la clave K mediante el HTC
$V[i] := [v_{[i,1]}, \ldots, v_{[i,16]}]$	salida de la S-caja en la ronda i, $i \in [1,4]$
$U[j] := [u_{[j,1]}, \ldots, u_{[j,16]}]$	entrada a la ronda j, $j \in [2,4]$ (después de sumar $V[j-1]$ con la clave)
$S(X)$	bloque binario resultante de aplicarle a X la sustitución por las S-cajas
$T(X)$	bloque binario resultante de aplicarle a X la transposición en sus bits

Entonces, asumiendo que P y C son conocidos, la modelación polinómica del cifrado HTC viene dada por la unión de los siguientes conjuntos de ecuaciones:

$F := \{k_h^2 + k_h = 0, v_{[i,h]}^2 + v_{[i,h]} = 0, u_{[j,h]}^2 + u_{[j,h]} = 0 \,|\, h \in [1,16], i \in [1,4], j \in [2,4]\}$,

$$H := \left\{ \begin{array}{ll} S(P+K) = V[1], & T(V[1]) + K = U[2], \\ S(U[2]) = V[2], & T(V[2]) + K = U[3], \\ S(U[3]) = V[3], & T(V[3]) + K = U[4], \\ S(U[4]) = V[4], & V[4] + K = C \end{array} \right\}.$$

Tabla 2. Entradas y salidas de la S-caja

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

F representa las ecuaciones del campo y H las del cifrado. F determina $16+4\cdot 16+3\cdot 16=128$ ecuaciones binarias, mientras que H da lugar a 128 ecuaciones binarias, para un total de 256 ecuaciones con 128 incógnitas (la clave y las variables de estado). De modo general, si se cifran m textos claros con la misma clave, entonces se formará un sistema con $16+m\cdot(4\cdot 16+3\cdot 16)=16+m\cdot 112$ ecuaciones del campo y $m\cdot 128$ ecuaciones del cifrado, para un total de $16+m\cdot 240$ ecuaciones y $16+m\cdot 112$ incógnitas.

En la literatura pueden encontrarse otros ejemplos de modelación polinómica de cifrados en bloques, ver, por ejemplo, [11, 1, 2].

Experimentación de HybridSolving para el HTC

La implementación del HybridSolving se realizó en la versión 17 de MAPLE. Los experimentos se llevaron a cabo en una PC con las siguientes características:

Intel Core i7-4790, a 3.6 GHz, con 16 GB de RAM.

A continuación se muestran varias tablas que recogen los resultados experimentales de la utilización de HybridSolving para el sistema de ecuaciones modelado a partir del cifrado HTC. La Tabla 4 muestra los tiempos de corrida en segundos y minutos (con aproximación hasta la centésima), al fijar $k=16,12,8$ y 4 variables de la clave respectivamente. Notemos que, en el caso de fijar las 16 componentes de la clave, no es posible obtener los resultados en breve tiempo (alrededor de una hora con 40 minutos).

Tabla 4. Tiempo de corrida al recorrer todo el espacio de los 2^k bloques binarios

k	2^k	Tiempo (segundos)	Tiempo (minutos)
16	65536	5982.42	99.70
12	4096	680.41	11.34
8	256	279.89	4.66
4	16	1303.37	21.72

Las columnas 3 y 4 de la Tabla 4 muestran que, al fijar sólo 4 variables, independiente de que sólo se les halle la solución a 16 sistemas, la estructura de estos es más compleja que en el resto de lo casos, en los cuales se fijan 2, 3 y 4 veces esta cantidad de variables y, por tanto, se disminuye más la cantidad de incógnitas del sistema inicial. Es decir, fijando 4 variables en el sistema inicial, es mayor la complejidad de la resolución de cada uno de los sistemas que se obtienen que la búsqueda exhaustiva.

Por otra parte, se observa en este caso que en $\frac{k}{2}$ se obtiene el mejor resultado, lo cual se asocia de manera interesante con lo declarado al inicio de la sección , sobre el compromiso entre la cantidad de variables que se fije y la complejidad y cantidad de sistemas de ecuaciones determinados por las variables fijadas. En este caso, la solución de mejor compromiso se alcanza justo en el medio.

Tabla 5. Tiempo de corrida al particionar en dos el espacio de los 2^k bloques binarios

c	Partición	Tiempo (segundos)
16	B_1	31.85
	B_2	31.01
12	B_1	1.71
	B_2	1.73
8	B_1	0.15
	B_2	0.12
4	B_1	0.03
	B_2	0.01

Una manera de simplificar el trabajo de la función HybridSolving es particionar el conjunto de bloques binarios que debe recorrerse y aplicar la función de modo independiente en cada uno de los subconjuntos de la partición. Este procedimiento se ajusta bien para la utilización de la computación en paralelo. El recorrido se estableció en los siguientes intervalos $[0,2^{k-1}]$ (subconjunto B_1) y $[2^{k-1}+1,2^k-1]$ (subconjunto B_2).

La columna 3 de la Tabla 5, arroja resultados alentadores: al particionar el conjunto, se evidencia una disminución en los tiempos de corrida en cada caso. Por construcción de los subconjuntos y ejemplo específico de clave, la misma siempre se encuentra en el mismo subconjunto de la partición (B_1), por lo cual, el tiempo para B_1 es ligeramente superior al tiempo para B_2, salvo en el caso en que se fijan 12 variables, por una diferencia muy pequeña.

3. Conclusiones

Se logra una implementación en GAP del Algoritmo F5 para el cálculo de bases de Gröbner. Teniendo en cuenta que F5 es uno de los mejores algoritmos para el cálculo de **BG** fue necesario consultar varias referencias, hasta encontrar un enfoque novedoso y unificado del mismo. Se profundiza en la metodología para el CA, basándose en literatura actual sobre el tema e incorporando enfoques y ejemplos desarrollados en este trabajo.

Se describe un ataque algebraico al cifrado en bloques HTC, en el cual se utiliza parte de la metodología mencionada en el párrafo anterior, así como se adicionan otras técnicas del

CA.

Se contribuye a esclarecer la metodología para la obtención de la modelación polinómica de los cifrados en bloques, en particular de las redes de substitución permutación, y se expone una implementación práctica de un ataque por enfoque híbrido, utilizando el sistema MAPLE, para ello fue necesario construir nuevas funciones en este sistema. Se obtienen resultados alentadores. En el intento de extenderlos a cifrados de mayor fortaleza, la utilización potencial de la computación paralela resulta también promisoria. Este trabajo puede ser útil además como otra referencia metodológica para el ataque algebraico, con la particularidad de que explica todo el proceso, desde la modelación teórica hasta su implementación práctica.

Referencias

[1] Albrecht, Martin: *Algorithmic algebraic techniques and their application to block cipher cryptanalysis*. Tesis de Doctorado, Citeseer, 2010.

[2] Bard, Gregory: *Algebraic cryptanalysis*. Springer Science & Business Media, 2009.

[3] Bettale, Luk, Jean Charles Faugere y Ludovic Perret: *Hybrid approach for solving multivariate systems over finite fields*. Journal of Mathematical Cryptology, 3(3):177–197, 2009.

[4] Bettale, Luk, Jean Charles Faugère y Ludovic Perret: *Hybrid approach: a tool for multivariate cryptography*. En *Proceedings of the ECRYPT Workshop on Tools for Cryptanalysis*, páginas 15–23, 2010.

[5] Bettale, Luk, Jean Charles Faugère y Ludovic Perret: *Solving polynomial systems over finite fields: improved analysis of the hybrid approach*. En *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, páginas 67–74. ACM, 2012.

[6] Faugère, Jean Charles: *A new efficient algorithm for computing Gröbner bases without reduction to zero (F 5)*. En *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, páginas 75–83. ACM, 2002.

[7] Ferrer, I. Martínez: *Criptoanálisis algebraico a cifrados en bloques*. Tesis en opción al título de Máster en Matemática. Universidad de La Habana, Cuba, 2018.

[8] Heys, Howard M: *A tutorial on linear and differential cryptanalysis*. Cryptologia, 26(3):189–221, 2002.

[9] M. Borges-Quintana, M.A. Borges-Trenard, I. Martínez Ferrer: *Algoritmo F5 en GAP, útil herramienta para la CCP*. En *Congreso Internacional Compumat 2015. La Habana, Cuba*, 2015.

[10] M. Borges-Quintana, M.A. Borges-Trenard, I. Martínez Ferrer: *Ataque Algebraico por Enfoque Híbrido a Cifrados en Bloques*. En *Congreso Internacional Compumat 2017. La Habana, Cuba*, 2017.

[11] Pellikaan, Ruud, Xin Wen Wu, Stanislav Bulygin y Relinde Jurrius: *Codes, cryptology and curves with computer algebra*. Cambridge University Press, 2017.

[12] Pub, NIST FIPS: *Announcing the advanced encryption standard (AES)*. Federal Information Processing Standards Publication, 197:1–51, 2001.

[13] Sun, Yao y Dingkang Wang: *The F5 algorithm in Buchberger's style*. Journal of Systems Science and Complexity, 24(6):1218–1231, 2011.