Ciencias Matemáticas, Vol. 35, No. 1, 2021, Pag. 21-26
Recibido 11-2020, Aceptado 04-2021

Acerca del criptoanálisis a cifrados en bloques mediante el Algoritmo Genético
On cryptanalysis of block ciphers by means of the Genetic algorithm

Miguel A. Borges Trenard[1], Mijail Borges Quintana[2], Adrian Donatien Charón[3]*

Resumen Con el presente trabajo continuamos investigando el efecto del Algoritmo Genético en el ataque a cifrados en bloques, basado en un particionamiento del espacio de claves. Se comparan las técnicas empleadas con respecto a un trabajo preliminar en esta misma dirección y se muestran los avances obtenidos con esas técnicas.

Abstract With the present work we continue investigating the effect of the Genetic Algorithm in the attack on block ciphers, based on a partitioning of the key space. The techniques used are compared with a preliminary work in this same direction and the advances obtained with these techniques are shown.

Palabras Clave
criptoanálisis, cifrado en bloque, algoritmo genético

Keywords
Cryptanalyisis,Block ciphers, Genetic algorithm

[1] *Departamento de Educación en Matemática, Universidad Antonio Nariño, Bogota, Colombia, miguelb@gmail.com*
[2] *Departamento de Matemática, Universidad de Oriente,Santiago de Cuba, Cuba, miajil@uo.edu.cu*
[3] *Departamento de Matemática, Universidad de Oriente,Santiago de Cuba,Cuba, adriand@uo.edu.cu*
***Autor para Correspondencia**

Introducción

El algoritmo genético se ha utilizado en criptoanálisis. Describamos brevemente algunos avances en esa dirección: En [12], los autores presentan una combinación de Algoritmo Genético con Optimización de Enjambre de Partículas (otro método heurístico basado en técnicas evolutivas), llamaron a su método Optimización de Enjambre Genético y lo aplicaron para atacar el DES . Sus resultados experimentales les muestran que se obtienen mejores resultados aplicando su método combinado que utilizando ambos métodos por separado.[5] proporciona una exploración preliminar del uso de *AG* en un cifrado de Red de Permutación de Sustitución (SPN), el propósito de la exploración es determinar cómo encontrar claves débiles.

Ambos artículos ([12] y [5]) utilizan un ataque a texto plano conocido, es decir, dado un texto plano T y el correspondiente texto cifrado C, el objetivo es encontrar la clave K. En [12], la función de aptitud evalúa la diferencia bit a bit (distancia de Hamming) entre C y el texto cifrado de T utilizando la clave candidata, mientras que, por el contrario, en [5] mide la distancia de Hamming entre T y el texto descifrado de C utilizando la clave bajo observación. [6] muestra un ataque sólo a texto cifrado al SDES, obteniendo mejores resultados que la fuerza bruta. Los autores utilizan una función de aptitud que es una combinación de la frecuencia relativa de monogramas, diagramas y trigramas (para un idioma en particular).

Como la longitud de la clave es muy pequeña, pueden utilizar este tipo de función. [1] es similar a [6], se usa la misma función de aptitud en esencia pero con diferentes parámetros, también es más detallado en cuanto a experimentos y se compara no solo con respecto a la fuerza bruta sino también a la búsqueda aleatoria.

[10] discute el uso de *AG* para el criptoanálisis al DES con el fin de mejorar el ataque tanto para el ataque diferencial como para el lineal. [7] presenta un ataque a ocho rondas del DES para obtener la clave exacta. Su método es una combinación de *AG* con criptoanálisis diferencial. Los autores dicen que el rendimiento de su ataque es considerablemente más rápido que la búsqueda exhaustiva y el criptoanálisis diferencial, además, el algoritmo se puede aplicar a una variedad de sistemas similares a DES, en lugar de las técnicas de criptoanálisis diferencial puro disponibles.

[2] es una breve descripción de lo que se ha hecho en el campo del criptoanálisis mediante el AG en los últimos 15 años. [8] también es una revisión del *AG* aplicado al criptoanálisis y también presenta un estudio de las bases de datos donde se puede encontrar este tema.

El lector no familiarizado con el *AG* puede ver una breve pero suficiente introducción en [3]. En este artículo se muestra un ataque general para cifrados en bloques por medio del

AG. Este es un ataque de caja negra, es decir, el intruso no necesita saber nada sobre la estructura del cifrado de bloques. El adversario solo necesita tener acceso a la maquinaria de cifrado y descifrado del cifrado en bloque, junto con al menos un par de texto plano y su correspondiente texto cifrado, es decir, es un ataque de texto plano conocido.

Como es conocido, el espacio de claves de un cifrado en bloques real es muy grande, para evitar un ataque de fuerza bruta, por lo tanto, el *AG* tiene que enfrentarse con la dificultad de buscar en este espacio y, aunque la búsqueda de *AG* es heurística, el tamaño del espacio de claves debe tenerse en cuenta. En [4] se realiza una partición del espacio de claves en base a una cierta congruencia aritmética, de modo que se pueda enfocar el ataque solo en algunos de los subconjuntos de la partición, por lo tanto, de esta manera el conjunto de claves por investigar es más pequeño.

La idea de utilizar la congruencia aritmética en el criptoanálisis parece interesante porque permite abordar problemas complejos con herramientas sencillas. Esta técnica también se utiliza en otros trabajos, por ejemplo, se aplica en [11], donde el autor demuestra que un determinado protocolo de intercambio de claves públicas se puede romper con un ataque man-in-the-middle, que se basa en (en principio) al trabajar con congruencias aritméticas.

En este trabajo se perfecciona el enfoque utilizado en [4] con el objetivo de incrementar el tamaño de los conjuntos que integran la partición y de esa forma ampliar la búsqueda en un espacio mayor. En la Sección 1, el lector puede ver una breve introducción al AG y al cifrado de bloques que elegimos para los experimentos.En la Sección 2, damos las especificaciones de la AG para nuestro ataque. La Sección 3, está dedicada a describir los resultados de nuestro método sobre el cifrado de bloque bajo ataque.

1. Preliminares

1.1 El Algoritmo Genético

Suponemos que el lector está familiarizado con algún conocimiento básico del algoritmo genético, por lo tanto, esta sección será breve, tratando de ser mínimamente autónoma.

El *AG* es una búsqueda aleatoria guiada que trabaja con poblaciones de individuos que son soluciones factibles para un problema dado. La evolución de tales poblaciones se basa en metáforas biológicas, cuando están sujetas a operadores probabilísticos.

El *AG* simula la evolución natural, imitando procesos como la selección, el apareamiento y la mutación.También simula la supervivencia de los más aptos entre los individuos

Algorithm 1 Genetic Algorithm.

1: Initialize population with random candidate solution:
2: **while** termination condition is not satisfied **do**
3: **Select** parents:
4: **Combine** pairs of parents:
5: **Mutate** the resulting offspring:
6: **Select** individuals of the next generation:
7: **end while**
8: **return** The individual with greater fitness.

durante generaciones consecutivas.

Por tanto, para aplicar el *AG* a un problema específico, uno tiene que definir con precisión (para el problema dado) los componentes y operadores de este algoritmo, a saber: cuáles son los individuos, cuántos individuos tiene una población, cómo los operadores de el trabajo de emparejamiento y mutación, cómo medir la aptitud de los individuos, cuáles son los criterios de selección para los individuos y cuál es la condición de terminación.

El último generalmente depende de los límites para el valor de aptitud o el número de generaciones. La sección 3.1 especializa el *AG* para el problema que se estudia en este artículo.

1.2 Familia de cifrado en Bloque AES(t)

En esta sección presentamos un cifrado que se denomina $AES(t)$, porque es una familia de cifrado en bloque bastante similar al AES, el cifrado depende de un parámetro t en un sentido que se aclarará más adelante. Esta familia fue introducida en [9]. A continuación, damos una descripción de la misma.

Sea el campo $\mathrm{GF}(2)$ de Galois con 2^t elementos dados por los polinomios de la siguiente tabla:

t (bits)	$m(x)$	block length	key length
3	x^3+x+1	48	48, 72, 96
4	x^4+x+1	64	64, 96, 128
5	$x^5+x^4+x^3+$ $+x^2+1$	80	80, 120, 160
6	$x^6+x^5+x^3+$ $+x^2+1$	96	96, 144, 192
7	$x^7+x^6+x^5+$ $+x^4+x^3+$ $+x^2+1$	112	112, 168, 224
8	$x^8+x^4+x^3+$ $+x+1$	128	128, 192, 256

En esa tabla, también se describe la longitud de bloque y clave de los miembros de t, para cada valor de t, donde t es la longitud de palabra correspondiente. En particular, AES es $AES(8)$. Todos los polinomios $m(s)$ son irreductibles sobre $GF(2)$, como dijo el autor en [9], estos polinomios fueron elegidos al azar. Por otro lado, los recuadros S para cada t se enumeran en el apéndice de ese documento.

Las operaciones de ronda de $AES(t)$ AddRoundKey, SubBytes, ShiftRows y MixColumns son en esencia las mismas que en AES, pero en una escala reducida. En particular, para MixColumns, los coeficientes de las matrices MDS son los t bits menos significativos de los coeficientes correspondientes de la matriz AES original. El mismo enfoque se aplica a las constantes utilizadas en la programación de claves. El número de rondas es el mismo que se utiliza en AES, incluso para diferentes tamaños de clave.

2. Algortimo Genético para el criptoanálisis de cifrados en bloque

2.1 Epecificación del algoritmo

Para esta sección seguimos esencialmente [4].

Sea $E : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ un cifrado en bloque, T texto claro, K es la clave y C el correspondiente texto cifrado, $C = E(K,T)$. Se dice que K' es una clave consistente E,T y C si $C = E(K',T)$. Denotemos con $Cons_E$ el conjunto de claves consistentes con E,T y C. El problema que se intenta resolver en este trabajo es : **Dado** E,T y C, **calcular** una clave en $Cons_E$.

Por lo tanto, este es un ataque a texto plano conocido. Los individuos de las poblaciones son claves que son candidatas a ser claves consistentes. A continuación, proporcionamos aquí la especificación que establecemos de los parámetros generales de AG para este caso.

Criterio de emparejamiento: selección de torneo con tamaño de torneo 2 (cf. [3]).

Función de Aptitud: $F(K) = \frac{E(K,T)*C}{n}$, donde $B_1 * B_2$ sigifica el número igual de compònentes entre los bloques B_1 and B_2. En otras palabras, $B_1 * B_2 = n - d_H(B_1,B_2)$, donde d_H es la distancia de Hamming's . Por tanto, K_1 es mas adaptado K_2 si $F(K_1) > F(K_2)$.

Esta función tiene en cuenta la correlación entre el texto cifrado conocido y el texto cifrado generado por las claves aleatorias. Esta función de aptitud es utilizada en [12].

El **operador de mutación:** Intercambia los valores de los bits en tres componenetes aleatorias como máximo, aplicando repetidamente el operador de mutación de un punto tres veces.

El **operador de cruzamiento en dos puntos :** El operador de cruzamiento en dos puntos selecciona aleatoriamente dos componentes de los padres y luego intercambia los bits de los dos padres entre estos puntos, para producir dos nuevos hijos.

2.2 Partición del espacio de la clave

El espacio de la clave es demasiado grande para una búsqueda exhaustiva, también podría ser incluso para una forma de búsqueda heurística. Por lo tanto, podría ser interesante dividir el espacio de claves en subconjuntos para que se aplique el algoritmo genético cada vez en solo uno de los subconjuntos.

Para ello, en [4] se propone la siguiente metodología: es bien sabido que si la longitud de la clave es k_1 entonces el espacio clave K tiene cardinal 2^{k_1} y existe una correspondencia uno a uno entre K y los números entero en el intervalo $[0, 2^{k_1} - 1]$. Si se fija un número entero k_2, $(1 < k_2 \leq k_1)$, entonces el espacio clave puede ser representado con los números $q \cdot 2^{k_2} + r$, donde $q \in [0, 2^{k1-k2} - 1]$ y $r \in [0, 2^{k_2} - 1]$.

De esta forma el espacio de la clave se divide en bloques 2^{k1-k2} (determinados por los cocientes en el algoritmo de división dividiendo por 2^{k_2}) y, dentro de cada bloque, la clave correspondiente se determina por su posición en el bloque, que viene dada por el residual r.

Se llama k_2 *la longitud de la clave para movimiento* o *longitud de clave grupal*. Entonces el punto clave es moverse con el algoritmo genético solo en una de las particiones (dada q), que se denomina *el bloque de la partición*, pero para calcular la aptitud correspondiente en todo el espacio real, es decir, moverse alrededor de r pero para calcular idoneidad de las claves dada por $q \cdot 2^{k_2} + r$.

Se definen las siguientes funciones para la programación: num : $\mathbb{N} \times \mathbb{Z}^2_{\geq 0} \to \mathbb{Z}_{\geq 0}$ tal que

$(k_2, q, r) \to q \cdot 2^{k_2} + r$, db y bd son las funciones que convierten decimal a binario y viceversa (usaremos la notación big endian, es decir, el bit más significativo aparece a la izquierda y va hacia el dígito menos significativo, a la derecha), (a,b) genera un número entero en el intervalo $[a,b]$. No describiremos las funciones member y insertt, porque su comportamiento es obvio. Ahora explicaremos paso a paso aquellas partes del algoritmo que requieren moverse desde la partición donde estamos trabajando a todo el espacio de claves, para poder calcular la aptitud en ese espacio.

Como verá el lector en el algoritmo 2, bloques en R_1 tienen longitud k_2 todas las operaciones del algoritmo genético se realizan en bloques de longitud k_2 (cruzamiento y mutación), sin embargo, la comparación de aptitud se realiza en el espacio real (longitud de clave $k1$). Por otro lado, con respecto al Algoritmo 3, recordamos que la función de cruzamiento devuelve dos descendientes, esta es la razón de los dos si. El cruzamiento se realiza con respecto a $k2$, mientras que la aptitud se calcula con respecto a $k1$.

Algorithm 2 RP: Población Aleatoria.

Input: m (cantidad de individuos en la población), k_1, k_2, q, F (funció de aptitud), es decir, F es una función que $\{0,1\}^{k_1} \to [0,1]$. Es necesario que $2^{k_2} \geq m$.

Output: $[R_1, R_2]$, donde R_1 es una población de representaciones binarias (de longitud k_1), de números en la partición determinada por q, cuyos residuos módulo 2^{k_2} se generan aleatoriamente, mientras que R_2 contiene la aptitud de los elementos de R_1.

1: $i := 1$; $R_1 := []$; $R_2 := []$;
2: **while** $i \leq m$ **do**
3: $r := \text{rand}(0, 2^{k_2} - 1)$;
4: $c := \text{db}(r)$;
5: **if** $\text{nott}(\text{member}(c, R_1))$ **then**
6: $R_1 := \text{insertt}(c, R_1)$;
$R_2 := \text{insertt}(F(\text{db}(\text{num}(k_2, q, r))), R_2)$;
7: $i := i + 1$;
8: **end if**
9: **end while**
10: **return** $[R_1, R_2]$;

Finalmente, en el Algoritmo 4, la función mut muta cada elemento $S[i]$ de la población S, esta mutación se realiza como ya se estableció en la Sección 2,1 (caso n = 3). Observemos que la representación decimal de $S[i]$ está en $[0, 2^{k_2} - 1]$. Si el mutante p es nuevo, entonces su aptitud asociada se calcula pasando a db(num (k2, q, p)) .

Algorithm 3 Cruzamiento

Input: M (población real), S (lista con los individuos que se van a cruzar), s (cantidad de individuos seleccionado para cruzarse), k_1, k_2, q, *pairing* (función de cruzamiento), F (función de aptitud).

Output: M actualizado con nuevos descendientes y su aptitud.

1: $R_1 := M[1]$; $R_2 := M[2]$;
2: **for** $i = 1, \ldots, s$ **do**
3: **for** $j = i + 1, \ldots, s$ **do**
4: $p := \text{pairing}(S[i], S[j], k_2)$;
5: **if** $\text{nott}(\text{member}(p[1], R_1))$ **then**
6: $R_1 := \text{insertt}(p[1], R_1)$;
$R_2 := \text{insertt}(F(\text{db}(\text{num}(k_2, q, p[1]))), R_2)$;
7: **end if**
8: **if** $\text{nott}(\text{member}(p[2], R_1))$ **then**
9: $R_1 := \text{insertt}(p[2], R_1)$;
$R_2 := \text{insertt}(F(\text{db}(\text{num}(k_2, q, p[2]))), R_2)$;
10: **end if**
11: **end for**
12: **end for**
13: **return** $[R_1, R_2]$;

Algorithm 4 Mutación

Input: M (población real), P_m (probabilidad de mutación), n (número de mutaciones),
k_1, k_2, q, F (función de aptitud).

Output: M actualizado con nuevos mutantes y su aptitud.

1: $S := M[1]$; $R_1 := M[1]$;
$R_2 := M[2]$;
2: **for** $i = 1, \ldots, \text{Cardinal}(S)$ **do**
3: $p := \text{mut}(S[i], k_2, P_m, n)$;
4: **if** $\text{nott}(\text{member}(p, R_1))$ **then**
5: $R_1 := \text{insertt}(p, R_1)$;
$R_2 := \text{insertt}(F(\text{db}(\text{num}(k_2, q, p))), R_2)$;
6: **end if**
7: **end for**
8: **return** $[R_1, R_2]$;

Respecto a las técnicas empleadas en el trabajo [4], se tienen las siguientes mejoras al algoritmo: en la búsqueda de la población inicial, hay una fase primera que consiste en buscar exhaustivamente individuos que tengan aptitud lo más alta posible, la fundamentación para ello es la siguiente: el AG es un algoritmo heurístico de búsqueda exhaustiva, es decir, de todos modos tiene su por ciento de fuerza bruta. En esa primera fase se aplica pura fuerza bruta, con el propósito de levantar lo más posible las aptitudes de los individuos que formarán parte de la población inicial. Posteriormente, en la nueva población de la próxima iteración, se ordenan los elementos con respecto a su actitud y una pequeña porción de los individuos de la nueva población se toman de entre los que tienen mayor aptitud, el resto de la nueva población se selecciona por el Método de los Torneos, lo cual ya se hacia en [4]. De ese modo se garantiza que las aptitudes máximas de las nuevas poblaciones nunca desciendan, se mantengan iguales o superiores que en las generaciones anteriores. De la experimentación realizada en [4], se evidenció que, si no se tiene en cuenta este criterio helitista de selección, se puede caer en un comportamiento oscilatorio de aptitudes que aumentan o disminuyen en cada iteración, o bien, se hace estacionario.

3. Experimentos

Trabajamos con el cifrado AES(3) de longitud de clave $k1 = 48$. Se toma un tamaño de $m = 100$ para la población en cada iteración. Como probabilidades de cruzamiento y mutación se tomaron respectivamente 0.9 y 0.01. Fue utilizada la misma función de aptitud que en [4]. También fijamos como longitud de clave grupal $k2 = 24$, superior a la longitud 16 utilizada en [4], por lo tanto el espacio de claves posibles tiene cardinal superior (2^{24}). No obstante, en los experimentos se tomó sólo una pequeña porción de esa población para recorrer. Para la mayor comprensión de la aseveración anterior, es conveniente destacar que el producto del número de iteraciones o generaciones (g) por la cantidad de individuos que tiene la

población en cada iteración (m) es una cota superior para la cantidad de elementos que se recorre en una corrida completa del AG. Por tanto, si tomamos g como la parte entera de $\frac{p2^{k2}}{m}$, donde $p = 0{,}0004$, por lo que el número de generaciones sería 67 y estaremos muestreando 6700 individuos cuanto más, de los 2^{24} posibles.

Se realizaron un total de 8 experimentos o corridas, cada experimento (clave y texto fijo) se repitió 10 veces. Luego de cada experimento, se estimaron las componentes de la clave utilizada mediante un análisis de frecuencia relativa simple. Un método algo semejante fue ya utilizado en [12]. El experimento se realizó con una computadora con las siguientes caracteristicas: Processor: Pentium(R) Dual-Core CPU T 4200 @ 2.00GHz

La siguiente tabla muestra las 8 componentes que coinciden con la clave que poseen mayor frecuencia relativa en cada corrida.

Corridas	Componente / Frecuencia
1	29(0,76), 42(0,71), 48(0,71), 46(0,66) 44(0,61), 43(0,61), 33(0,57), 37(0,52)
2	28(0,64), 26(0,57), 27(0,57), 31(0,57) 41(0,57), 42(0,57), 46(0,57), 30(0,57)
3	46(0,70), 30(0,70), 27(0,70), 48(0,64) 38(0,64), 41(0,58), 29(0,58), 33(0,58)
4	30(0,7), 47(0,66), 43(0,66), 46(0,62) 31(0,62), 27(0,62), 28(0,62), 44(0,58)
5	36(0,78), 46(0,68), 34(0,68), 48(0,63) 38(0,63), 30(0,63), 27(0,63), 35(0,57)
6	48(0,70), 45(0,70), 39(0,70), 46(0,60) 44(0,60), 42(0,60), 37(0,60), 27(0,60)
7	37(0,73), 43(0,68), 42(0,68), 46(0,58) 29(0,58), 27(0,58), 34(0,57), 33(0,57)
8	48(0,71), 46(0,71), 43(0,71), 36(0,64) 32(0,64), 29(0,52), 33(0,52), 37(0,52)

Los resultados se pueden resumir de la siguiente manera:

En cada corrida se estimó una clave por el análisis de frecuencia relativa simple.

En todas las corridas desde la posición 1 hasta la 24, las componentes coinciden. Esto es debido al bloque de la partición de longitud 24 con el que se está trabajando, todos los elementos de la población pertenecientes a este bloque tienen los mismos valores en las 24 primeras posiciones.

En cada corrida de las posiciones 25 a la 48 al menos hay 8 que coinciden con las de la clave.

Análisis de las componentes de mayor frecuencia

En cada clave estimada por frecuencia relativa (kef) de las corridas se eligieron las componentes con mayor frecuencia relativa desde la posición 25 hasta la 48. Esto lo hicimos para ver si las posiciones de mayor frecuencia relativa coinciden con la de la clave.

En la corrida 1 de las 8 que coinciden con la clave, 6 están dentro de la mayor frecuencia y ocupan las primeras posiciones: 1. (29) (0.76), 2. (48)(0.71), 3. (42) (0.71), 4. (46)(0.66).

frecuencia relativa mayor (frma):= 0,76

frecuencia relativa menor (frme) :=0,61

En la corrida 2, además de las 8 componentes que están en la tabla hay una componente más que coincide con la clave que es la 29, con frecuencia relativa de (0.5), ninguna de las componentes en esta corrida están dentro de las que tienen mayor frecuencia relativa. Un indicio que puede indicar que este caso no es confiable el análisis de predecir las componentes de la clave mediante las que tienen mayor frecuencia es el hecho de que las aptitudes en esas corridas son más bajas.

En la corrida 3 hay tres componentes más que no están en la tabla que coinciden con la clave $34\,(0{,}52)$, $39\,(0{,}52)$, $44\,(0{,}52)$. De las 11 que coinciden con la clave, 8 están dentro de las de mayor frecuencia. Las tres primeras componentes de la tabla en esta corrida coinciden con las de mayor frecuencia relativa. En esta corrida se encontró la clave.

frecuencia relativa mayor (frma):=0,7

frecuencia relativa menor (frme):=0,58

En la corrida 4 hay tres componentes que no están en la tabla que coinciden con la clave $36(0{,}54)$, $40(0{,}54)$, $42(0{,}58)$. De las 11 componentes que coinciden con clave, 7 están dentro de las de mayor frecuencia y están en las posiciones 30 (6.), 47 (7.), 43 (8.) y 46 (9.). Aunque las de mayor frecuencia relativa no coincidan con la clave, están en la posiciones $33\,(0{,}75)$ y $41\,(0{,}75)$. Nuevamente ocurre, como en la corrida 2, que las aptitudes asociadas a esta corrida son más bajas.

frecuencia relativa mayor (frma):=0,7.

frecuencia relativa menor (frme):=0,62.

En la corrida 5 hay 7 componentes que no están en la tabla que coinciden con la clave $31\,(0{,}52)$, $32\,(0{,}52)$, $39\,(0{,}52)$, $41\,(0{,}52)$, $44\,(0{,}52)$, $45\,(0{,}52)$, $47\,(0{,}52)$. De las 15 componentes que coinciden con la clave 7 están dentro de las de mayor frecuencia y están en las posiciones 4. (36), 5. (46), y 6.(34). Aunque las de mayor frecuencia relativa no coincidan con la clave están en las posiciones $29\,(0{,}84)$,,$25(0{,}84)$, $40(0{,}78)$, nuevamente las aptitudes en éstas corridas son más bajas.

frecuencia relativa mayor (frma):=0,78

frecuencia relativa menor (frme):=0,63

En la corrida 6 hay 6 componentes que no están en la tabla que coinciden con la clave $29\,(0{,}6)$, $33\,(0{,}6)$, $25\,(0{,}5)$, $26\,(0{,}5)$, $28\,(0{,}5)$, $34\,(0{,}5)$. De las 14 componentes que coinciden con la clave 10 están dentro de las de mayores frecuencias relativas, y coincide con la 1. (48) (0.70), en esta corrida se encontró la clave.

frecuencia relativa mayor (frma):=0,7

frecuencia relativa menor (frme):=0,6

En la corrida 7 hay 6 componentes que no están en la tabla que coinciden con la clave 26 (0,57), 28(0,57), 30 (0,57), 36 (0,52), 38 (0,52), 45 (0,52). De las 14 componentes que coinciden con la clave 11 están dentro de las de mayores frecuencias relativas y coincide con la 1. (37) (0.73).

frecuencia relativa mayor (frma):=0,73

frecuencia relativa menor (frme):=0,57

En la corrida 8 hay 1 componente que no esta en la tabla que coincide con la clave 40 (0,52). De las 9 componentes que coinciden con la clave 5 están dentro de las de mayores frecuencias y coinciden con la 1. (48) (0.71), 2. (46)(0.71), 3. (43)(0.68).

frecuencia relativa mayor (frma):=0,71

frecuencia relativa menor (frme):=0,64

En la tercera y sexta corrida se encontró la clave. En las corridas donde se encontró la clave las frecuencias relativas son más altas.

En todas las corridas hay al menos una componente que está dentro de las que tienen las frecuencias relativas más altas. En todas las corridas salvo en la segunda, la componente 46 se repite. En la mayoría de las corridas mientras más alta es la frecuencia relativa de esa componente mayor probabilidad tiene de coincidir con la de la clave.

El Promedio de frecuencia relativa mayor es de 0,70875

El Promedio de frecuencia relativa menor es de 0,60712

4. Conclusiones

Las técnicas introducidas en este trabajo, con respecto al método empleado en [4], han conducido al incremento de las potencialidades del AG para el ataque a cifrados en bloques, por el método de particionamiento del espacio de claves, teniendo en cuenta que se incrementa la longitud del espacio de búsqueda y en varias corridas se puede obtener la clave y en general se describen varios aspectos positivos observados en los experimentos que permite obtener información sobre la clave.

Como es habitual en las aplicaciones de *AG*, se podrían obtener algunas mejoras de los resultados si se consideran algunas variaciones en los parámetros, a saber, operadores de cruzamiento y mutación (junto con sus probabilidades), número de individuos en la población inicial, número de generaciones, y similares.

Una observación aparte merece la función de aptitud, podría ser interesante comparar los resultados usando la función definida en [5], en lugar de la que hemos usado (que apareció en [12]). Un inconveniente que tiene el método es que cuanto menor es $k2$ resulta mayor el número de particiones (2^{k1-k2} bloques). Por otra parte, el incremento de $k2$ significa ampliar la longitud del espacio de búsqueda. Por lo que es necesario encontrar un compromiso entre ambos criterios. Este trabajo contribuye a la obtención de mayor información cuando se incrementa $k2$ que permita realizar búsquedas posteriores reduciendo nuevamente el espacio de búsqueda haciendo uso de la información obtenida. Este ataque podría ser especialmente adecuado para los casos en los que se considera que algunos bits de las posibles claves tienen valores fijos; en este caso, el problema se puede reducir al ataque propuesto por *AG* sobre el resto de las posiciones desconocidas.

Referencias

[1] F. Al Adwan, M. Al Shraideh, and M. R. Al Saidat: *A genetic algorithm approach for breaking of simplified data encryption standard*. International Journal of Security and Its Applications, 2015.

[2] S. Baragada and P. S. Reddy: *A survey of cryptanalytic works based on genetic algorithms*. International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 2013

[3] R. K. Bhattacharjya: *Introduction to genetic algorithms*. IIT Guwahati, 2012

[4] M. A. Borges-Trenard, M. Borges-Quintana, L. Monier-Columbié: An application of genetic algorithm to cryptanalysis of block ciphers by partitioning the key space. Journal of Discrete Mathematical Sciences & Cryptography, 2019. DOI: 10.1080/09720529.2019.1649028.

[5] J. A. Brown, S. Houghten, and B. Ombuki-Berman: *Genetic algorithm cryptanalysis of a substitution permutation network*. Computational Intelligence in Cyber Security, 2009

[6] P. Garg, S. Varshney, and M. Bhardwaj: *Cryptanalysis of simplified data encryption standard using genetic algorithm*. American Journal of Networks and Communications, 2015

[7] H. M. H. Husein, B. I. Bayoumi, F. S. Holail, B. E. M. Hasan, and M. Z. A. El-Mageed: *A genetic algorithm for cryptanalysis of des-8*. IJ Network Security, 2007

[8] A. H. Khan, A. H. Lone, and F. A. Badroo: *The applicability of genetic algorithm in cryptanalysis: A survey* International Journal of Computer Applications, 2015.

[9] J. Nakahara and D. Santana de Freitas: *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[10] T. Tadros, A. E. F. Hegazy, and A. Badr: *Genetic algorithm for des cryptanalysis genetic algorithm for des cryptanalysis*. Int. J. Comput. Sci. Netw. Secur, 2010.

[11] M. R. Valluri: *Cryptanalysis of Xinyu et al.'s NTRU-lattice based key exchange protocol*. Journal of Information and Optimization Sciences, 2018.

[12] R. Vimalathithan and M. Valarmathi: *Cryptanalysis of simplified-des using computational intelligence*. Transactions on Computers, 2011.