Ciencias Matemáticas, Vol. 31, No. 1, Pag. 61-65, 2017
Recibido 02-2017

Algoritmo Genético en el Criptoanálisis a Cifrados en Bloques
Genetic Algorithm in Cryptanalysis of Block Ciphers

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

Resumen Es conocido que el Algoritmo Genético ha sido aplicado al criptoanálisis de cifrados en bloques. Aquí se propone una alternativa a estos ataques y una metodología de trabajo.

Abstract It is known that the Genetic Algorithm has been applied to the criptanalysis of block ciphers. Here it is proposed an alternative to these attacks and a methodology of work.

Palabras Clave
Cifrados en bloques, criptoanálisis, algoritmo genético

[1] *Departamento de Matemática, FCNE, Universidad de Oriente, Cuba.*
mijail@uo.edu.cu, mborges@uo.edu.cu, adriand@uo.edu.cu.
***Autor para Correspondencia**

1. Introducción

Proteger la información sensible es una necesidad insoslayable de la sociedad moderna. Del desarrollo de herramientas matemáticas y computacionales para lograr ese propósito se ocupan disciplinas científicas como la Criptografía, a través de los denominados protocolos criptográficos. Los cifrados en bloques constituyen una de las primitivas criptográficas más utilizadas en esos protocolos.

El Algoritmo Genético (AG) se ha utilizado en el criptoanálisis a cifrados en bloques. Por ejemplo, en [11] se combina el AG con otro algoritmo evolutivo (Optimización de Enjambres de Partículas), con el propósito de realizar un ataque (a texto claro conocido) al conocido cifrado en bloques DES. Al parecer, con una población fija de 1000 individuos, realizando 500 ejecuciones en cada experimento, y repitiendo 3 veces el algoritmo, los autores dicen obtener una solución. En [5] se combina el **AG** con técnicas del criptoanálisis diferencial para atacar cifrados en bloques con estructuras semejantes a las del DES, en particular lo aplican al DES y FEAL con 8 rondas. Los autores declaran que el problema de utilizar un gran número de pares "correctos"(de texto claro con su cifrado) se puede resolver generando los pares correctos genéticamente, obteniendo un método más rápido que el diferencial. En [3] se trata sobre el problema de determinar si un cifrado determinado produce o no salidas aleatorias. Varias de las técnicas de criptoanálisis se basan en el hecho de que los cifrados bajo estudio no producen salidas verdaderamente aleatorias. Los autores abordan ese problema utilizando el **AG** y afirman obtener distinguidores para la no aleatoriedad más rápidos que otros reportados antes en la literatura, aplicando en particular el ataque a la Red de Feistel denominada TEA (Tiny Encryption Algorithm), con menos de 3 rondas. En el presente reporte proponemos una función de adaptación diferente a la utilizada en [11] y fundamentamos su ventaja. Ilustramos cómo realizar un ataque a texto claro conocido a un cifrado en bloques arbitrario y ejemplificamos con el cifrado HTC.

El trabajo se compone de las siguientes secciones: En la Sección 2 se presenta las especificaciones del **AG** para el caso bajo estudio (asumimos un lector conocedor de los elementos básicos de los algoritmos genéticos). La Sección 3 ilustra lo presentado en la Sección 2, con el criptoanálisis al cifrado HTC y presenta una nueva función de aptitud. Finalmente las conclusiones.

2. AG para el ataque a cifrados en bloques binarios

2.1 Algoritmo Genético

Para describir el Algoritmo Genético, nos basaremos en el esquema que se describe en [9]:

1. Inicializar las variables del algoritmo:
   g (número máximo de generaciones a considerar),
   M (tamaño de la población).
2. Generar una población inicial **P** con **M** individuos.
3. Calcular las aptitudes de los individuos en **P**.
4. Para **g** iteraciones realizar lo siguiente:
   a) Seleccionar parejas de progenitores en **P**.
   b) Parear los progenitores seleccionados.
   c) Para cada pareja generar un par de descendientes, usando la función de apareamiento seleccionada.

d) Aplicar una mutuación a cada uno de los descendientes.

e) Calcular las aptitudes de cada uno de los descendientes y sus mutaciones.

f) Basado en las adecuaciones de los progenitores y descendientes, decidir cuál será la nueva población **P**.

5. Seleccionar el individuo de mejor aptitud de la última generación como la mejor solución.

Un estudio básico pero suficiente para nuestros propósitos del **AG** puede lograrse en [4], [8], [10].

2.2 Cifrador en bloques binario

Para este contexto, un bloque binario es un objeto de la forma $[x_1,\ldots,x_p]$, donde $x_i \in \{0,1\}$, para $i = 1,\ldots p$. Se dice que la longitud del bloque es p. El conjunto de todos los bloques binarios de longitud p será denotado por $\{0,1\}^p$.

Un cifrado en bloques (binario) es una aplicación $E : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ tal que, para cada $K \in \{0,1\}^m$, $E(K,T)$ es una función inversible (función de cifrado en K). Los bloques $T \in \{0,1\}^n$ se denominan textos claros, mientras que $C = E(K,T)$ es el texto cifrado de T correspondiente a la clave K. También se dice que K es consistente con (T,C), con respecto al cifrado E. La función inversa $D(K,C)$ es la correspondiente función de descifrado. El número n es la longitud de bloque del cifrado y m es su longitud de clave. El espacio de las claves, $\{0,1\}^m$, debe ser lo suficientemente grande como para que el ataque por fuerza bruta no sea factible.

La literatura sobre los cifrados en bloques es amplia, sugerimos consultar [1].

2.3 Características del AG para el ataque a cifrados en bloques

Poblaciones: subconjuntos del espacio de claves.
Método para el criptoanálisis: El tipo de ataque es "a texto claro conocido", en el que se conoce un par (T,C) de texto claro T y su cifrado C y se busca K tal que $E(K,T) = C$, es decir, se busca una clave consistente.
Función de aptitud (sugerida en [11]): Es una aplicación A_1 de $\{0,1\}^m$ en $[0,1]$ definida por

$$A_1(K) = \frac{E(K,T) * C}{n},$$

donde $B_1 * B_2$ representa el número de componentes iguales entre los bloques B_1 y B_2. En otras palabras, $B_1 * B_2 = n - d_H(B_1,B_2)$, donde d_H es la distancia de Hamming.
K_1 es más adaptada que K_2 si $A_1(K_1) > A_1(K_2)$. Esta función tiene en cuenta la correlación entre el texto cifrado conocido y los textos cifrados generados por claves aleatorias.
Operador de Cruzamiento: Cruzamiento por un punto (consultar referencias citadas sobre el **AG**).
Operador de Mutación: Consiste en hallarle el complemento a dos componentes del bloque sometido a mutación (seleccionadas aleatoriamente). Ejemplo, una mutación de $[0,1,1,1,0,1,0,0]$ pudiese ser $[0,\underline{0},1,1,0,1,0,\underline{1}]$.

3. Ataque al cifrado en bloques HTC

3.1 Ataque al cifrado en bloque HTC

En esta sección aplicaremos el método de ataque explicado en la sección anterior al cifrado en bloques HTC. Este cifrado fue introducido en [6], sus parámetros son moderados y el propósito de Heys al presentarlo fue el de ilustrar cómo realizar los ataques lineal y diferencial a un cifrado en bloques. Aquí también lo utilizaremos con un propósito semejante, nuestro objetivo es ilustrar el método de trabajo y la función de adaptación que proponemos, para ello no es necesario utilizar un cifrado con mayor fortaleza, en cierto sentido es preferible utilizar uno como el HTC, de fácil comprensión e implementación, de modo que el lector interesado pueda seguir con facilidad lo esencial.

3.2 Descripción del HTC

HTC es un cifrador con las características siguientes: $m = n = 16$. Se repite una sucesión de pasos que abajo se describen (4 veces), por ello se dice que es un cifrador con $r = 4$ rondas. Cada ronda consiste de: una suma Xor con la clave de ronda, una substitución, y una transposición. No obstante, la última ronda tiene ciertas diferencias. En las rondas intervienen las siguientes permutaciones:

Substitución: El texto que se está cifrando se divide en 4 subbloques de 4 bits. Cada sub-bloque forma una entrada para una S-caja de 4×4, es decir, una aplicación $S : \{0,1\}^4 \to \{0,1\}^4$ (en este caso se requiere que sea biyectiva). Su representación en el sistema hexadecimal viene dada a continuación:

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

A	B	C	D	E	F
6	C	5	9	0	7

Se cuenta también con una **transposición** de los 16 bits.

1	2	3	4	5	6	7	8	9	10
1	5	9	13	2	6	10	14	3	7

11	12	13	14	15	16
11	15	4	8	12	16

Cada bloque de entrada en cada ronda se suma con la correspondiente clave de ronda (son 5 claves de ronda, pertenecientes a $\{0,1\}^{16}$, una opción que se ha utilizado es tomarlas todas iguales).
Utilizaremos las siguientes notaciones:
Si $X = [x_1,x_2,\ldots,x_{16}] \in \{0,1\}^{16}$, entonces, $X[i,j]$, para $i < j$, denota el subbloque de X formado por las componentes $x_i,x_{i+1},\ldots,x_j$. Como es usual, el operador $\oplus$ representa la suma Xor, en este caso, la suma en el grupo aditivo $\mathbb{Z}_2^{16}$.

Cifrado HTC:

Entrada: $M_0 := [m_{0,1}, m_{0,2}, \ldots, m_{0,16}]$ (texto claro),

$K_1,\ K_2,\ K_3,\ K_4,\ K_5$ (claves).

Salida: C_0 (cifrado de M_0).

$M_r = [m_{r,1}, m_{r,2}, \ldots, m_{r,16}] = \rho_r(K_r, M_{r-1})$, $r = 1,2,3$,

donde ρ_r se define en tres pasos:

$M_r := [m_{r-1,1}, m_{r-1,2}, \ldots, m_{r-1,16}] \oplus [k_{r,1}, k_{r,2}, \ldots, k_{r,16}]$;

$M_r := [S(M_r[1,4]), S(M_r[5,8]), S(M_r[9,12]), S(M_r[13,16])]$;

$M_r := [m_{r,\tau(1)}, m_{r,\tau(2)}, \ldots, m_{r,\tau(16)}]$.

Última ronda:

$M_4 := [m_{3,1}, m_{3,2}, \ldots, m_{3,16}] \oplus [k_{4,1}, k_{4,2}, \ldots, k_{4,16}]$;

$M_4 := [S(M_4[1,4]), S(M_4[5,8]), S(M_4[9,12]), S(M_4[13,16])]$;

$C_0 := M_4 \oplus [k_{5,1}, k_{5,2}, \ldots, k_{5,16}]$.

Descifrado HTC (ahora la primera ronda es la diferente)**:**

Entrada:

$C_0 := [c_{0,1}, c_{0,2}, \ldots, c_{0,16}]$, K_5, K_4, K_3, K_2, K_1 (claves).

Salida: M_0.

$C_1 := C_0 \oplus [k_{5,1}, k_{5,2}, \ldots, k_{5,16}]$;

$C_1 := [S^{-1}(C_1[1,4]), S^{-1}(C_1[5,8]), S^{-1}(C_1[9,12]),$
$S^{-1}(C_1[13,16])]$;

$C_1 := C_1 \oplus [k_{4,1}, k_{4,2}, \ldots, k_{4,16}]$;

$C_{r+1} = [c_{r+1,1}, c_{r+1,2}, \ldots, c_{r+1,16}] := \rho_r^{-1}(K_{4-r}, C_r)$,

$r = 1,2,3$, donde ρ_r^{-1} se define como sigue:

$C_{r+1} := [c_{r,\tau^{-1}(1)}, c_{r,\tau^{-1}(2)}, \ldots, c_{r,\tau^{-1}(16)}]$;

$C_{r+1} := [S^{-1}(C_{r+1}[1,4]), S^{-1}(C_{r+1}[5,8]), S^{-1}(C_{r+1}[9,12]),$
$S^{-1}(C_{r+1}[13,16])]$;

$C_{r+1} := C_{r+1} \oplus [k_{4-r,1}, k_{4-r,2}, \ldots, k_{4-r,16}]$.

Finalmente, $M_0 = C_4$.

3.3 Panorámica de la implementación del ataque en Maple

Se elaboraron programas en el sistema de cálculo simbólico denominado Maple (Versión 17), capaces de ejecutar el ataque presentado en este trabajo. Aquí sólo describiremos la función principal (gen). Al lector familiar con Maple le será más fácil entender estas instrucciones, no obstante, las mismas pueden ser comprendidas en su esencia por uno menos conocedor. Preferimos escribir el programa en Maple, en lugar de seudocódigo, para ilustrar las potencialidades de este sistema en la realización de este ataque. Estamos en disposición de aclarar y brindar más información al lector interesado sobre el programa, los experimentos, y cualquier otro aspecto asociado con este trabajo.

Entrada: M: tamaño de la población, f: cifrador,
Par: función de selección de parejas, mut: operador de mutación, cru: operador de cruzamiento, P: población inicial,
T: texto claro, C: texto cifrado, k: longitud de clave, g: número de generaciones.

Salida: Una terna cuya primera componente es la generación en que se alcanzó la solución (individuo de mayor aptitud), la segunda es la clave que se obtiene como solución y la tercera es su aptitud.

gen := proc(M, f, Par, mut, cru, P, T, C, k, g)

local t, c, Q, i, j, P1, P2: t := db(T, k): c := db(C, k):

(1): Q := genA1(M, P, k, f, t, c):

(2): for i to g do

(3): P1 := apply(Par, M):

P1 := [seq([Q[P1[j][1]],Q[P1[j][2]]], j = 1..nops(P1))]:

(4): P1 := Flatten([seq(apply(cru, P1[j][1][1], P1[j][2][1], k), j = 1..nops(P1))], 1):

(5): P2 := P1:

(6): P1 := [seq([P1[j], A1(t, P1[j], c, f)], j = 1..nops(P1))]:

(7): P2 := mu(P2, mut, k):

P2 := [seq([P2[j], A1(t, P2[j], c, f)], j = 1..nops(P2))]:

(8): Q := [seq(Q[j], j = 1..nops(Q)),

seq(P1[j], j = 1..nops(P1)), seq(P2[j], j = 1..nops(P2))]:

(9): Q := convert(convert(Q, set), list):

(10): Q := sort(Q, ord): Q := [seq(Q[j], j = 1..M)]:

(11): if Q[1][2] = 1 then return [i - 1, Q[1][1], Q[1][2]] fi:

(12): od: [i - 1, Q[1][1], Q[1][2]]: end proc:

Comentarios:

1. Aunque el programa se experimentó con el cifrado HTC, el mismo puede ser en principio aplicado con cualquier otra función de cifrado.
2. Semejantes consideraciones son válidas para los argumentos Par, mut, y cru.
3. En Par se utilizó la forma más elemental de seleccionar parejas (la aleatoria, con todos los elementos con la misma probabilidad de ser seleccionados) y para mut y cru se utilizaron respectivamente el cruzamiento por un punto y la mutación en dos componentes.
4. Siempre se utilizó como población inicial una generada aleatoriamente por el programa.

Descripción de la función gen:

(1): La función genA1 (que no se detalla aquí), construye la población inicial (determinada por el usuario o generada aleatoriamente) y asigna a cada individuo su aptitud. genA1 depende de la función A_1 (definida en la Sección 2.3). En general, la función gen depende de A_1 (como puede observarse), no obstante, las modificaciones que habría que introducir para utilizar otra función de aptitud son sencillas. T y C se introducen en forma decimal (para facilitar la comunicación con el usuario), de ahí que la función db transforma de decimal a binario de longitud k.

(2): Definición del lazo en el número de generaciones (iteraciones).

(3): Selección de las parejas para el cruzamiento: La función Par que se utiliza forma todas las combinaciones posibles de M elementos tomados dos a dos. Este conjunto de pares sirve como índices para construir las parejas de elementos de la población inicial, todos con la misma probabilidad de ser seleccionados. La función apply es primitiva de Maple, consiste en aplicar la función que tiene como primer argumento a los objetos que aparecen como siguientes argumentos. También es una primitiva muy utilizada de Maple la función seq, que forma la sucesión de elementos generados por su primer argumento,

cuando el conjunto de índices varía en el rango considerado.

(4): Obtención de los descendientes. La función Flatten es primitiva de Maple y tiene el propósito de hacer lineal la estructura anidada que se forma con las instrucciones indicadas en este paso.

(5-7): Mutación de los descendientes y actualización de las aptitudes, tanto en los descendientes como en sus mutantes.

(8): Unión de todas las poblaciones: la inicial, la de los descendientes y sus mutaciones.

(9): Eliminación de los elementos redundantes en la población obtenida en el paso anterior. (Convierte un conjunto de elementos en una lista.)

(10): Ordenamiento de la población actual (en orden descendiente de las aptitudes) y selección de los primeros M elementos de la población ordenada. La primitiva de Maple sort ordena su primer argumento según el criterio determinado por su segundo argumento.

(11): Criterio de parada (intermedia) del programa: se alcanza una solución (individuo de aptitud uno) antes de llegar al número total de generaciones posibles.

(12): Fin del lazo y fin del programa.

3.4 Nueva función de aptitud

Con la función de aptitud utilizada en [11], se logra encontrar una clave consistente con el par (T,C) con respecto al cifrado E, pero ello no garantiza que se encuentre la clave exacta, es decir, la clave que realmente se utilizó para cifrar a T y obtener C, ésto se debe a que dos claves diferentes pudiesen cifrar igual al texto claro T. La función de aptitud A_1 mide sólo proximidad al cifrado C. Teniendo en cuenta lo anterior, hemos construído una función de aptitud A_2, que tiene en cuenta tanto la proximidad a C como la cercanía a la clave utilizada. A_2 se construye a partir de una función auxiliar A_0, la cual mide solamente proximidad entre claves.
Veamos los detalles: A_0 depende de una clave variable K, una clave constante K_1 y una lista u ordenada de números (no repetidos) del conjunto $\{1,2,\ldots,|K|\}$. Entonces,

$$A_0(K,u,K_1) = \frac{NCI(K,u,K_1)}{|u|},$$

donde $NCI(K,u,K_1)$ es el número de componentes iguales entre K y K_1 que son indexadas por u.

Como puede verse, A_0 no depende del cifrado, sólo mide la proximidad entre K y K_1 en el subconjunto de índices determinado por u. Es decir, ¿cuán cercana se encuentre una clave K a una clave buscada en que se han fijado como conocidos los valores de algunas de sus componentes? Partiendo de A_1 y A_0 construímos una función que tiene en cuenta las proximidades tanto al cifrado como a la clave que se usó:

$$A_2 = \frac{\alpha A_0 + \beta A_1}{2}, \text{ donde } \alpha,\beta \in \mathbb{R},\ \alpha+\beta = 1.$$

En este trabajo tomamos como pesos $\alpha = 0{,}9$, $\beta = 0{,}1$. Es decir, se le otorgó más peso a la aproximación a la clave real.

La idea de fijar algunos de las componentes de la clave incógnita ha sido utilizada en diversas técnicas de criptoanálisis. Por ejemplo, en la denominada estrategia de "Suponer y Determinar"(Guess-and-Determine), la cual se puede describir como una repetición de dos pasos, hasta que todas las componentes han sido determinadas: primero se suponen conocidos los valores de algunas de las componentes y segundo se determinan los valores de las restantes componentes (tantas como sea posible). En el segundo paso, puede que en lugar de obtenerse los valores de las no fijadas se obtengan relaciones simplificadas entre esas componentes o puede llegarse a alguna contradicción que indique que algunas de las fijadas deben modificarse (ver, por ejemplo, [2]). Otro escenario en el que se presenta la alternativa de fijar algunas de las componentes es en los ataques de canal colateral ([7]), que son aquellos en los que el intruso puede obtener información adicional sobre los datos del cifrado que se esté ejecutando, utilizando para ello (por ejemplo) propiedades físicas de la implementación.

3.5 Experimentos

Realizamos dos grupos de experimentos, el Grupo I fue con la función A_1 y el Grupo II con A_2. Fueron comunes y fijos a ambos grupos los siguientes parámetros: texto claro, clave, cantidad de elementos en la población (20), función de apareamiento natural (basada en selección aleatoria de parejas, donde todos los individuos de la población tienen la misma probabilidad de ser seleccionados), los operadores de cruzamiento y mutación fueron los mencionados en la Sección 2.3, se realizó una selección aleatoria de la población inicial, se llevaron a cabo 220 iteraciones. Cada grupo de experimentos se repitió 5 veces. A continuación relacionamos algunas conclusiones derivadas de las experimentaciones.

En el Grupo I, las soluciones obtenidas (K) en las 5 repeticiones del experimento dieron lugar a cifrados $E(T,K)$ que tenían 14 componentes como promedio iguales a los de C, mientras que las claves usadas eran iguales a la clave real sólo en 7 componentes como promedio.

En el Grupo II se observó que las aproximaciones al cifrado y a la clave real se comportaron de manera más homogénea y que la cantidad de componentes que se obtuvieron iguales con la clave real dependió de la cantidad de componentes que se fijaron como conocidas (como era de esperar). La relación entre el número de componentes fijadas y la mejor aproximación a la solución no es del todo lineal, pero bastante aceptable. La siguiente tabla ilustra los resultados, en la misma, CI representa el número de componentes fijadas de la clave real, CII es el promedio (en los cinco experimentos) de componentes iguales con la clave real y CIII con el cifrado. No se incluye en la tabla los valores para los casos en que el número de componentes que se fija es mayor que 6 porque de ese valor en adelante se alcanza que CII = CIII = 16.

CI	CII	CIII
1	7	14
2	9	15
3	8	14
4	8	14
5	14	15

De la tabla anterior se observa que fijando entre 2 y 5 componentes, ya se obtiene el 50 % o más de las componentes de la clave real. Por otra parte, si se fijan 6 componentes o más, o sea, el 38 % de las componentes de la clave real, ya se obtiene ésta. Es también interesante destacar la cantidad total de individuos que se muestrean para llegar a buenas soluciones. Para ello observemos que en cada iteración se le aplica el cifrado a 60 individuos, contando la población de inicio, los apareamientos y las mutaciones (suponiendo que en esas operaciones no se repite ningún individuo) y se hacen 220 repeticiones, quiere decir que se exploran 13200 llaves posibles cuanto más, del total de 2^{16} (262144), lo cual representa sólo un 20 %.

4. Conclusiones

1. Hemos presentado una propuesta de especificación del algoritmo genético (Sección 2.1) para el criptoanálisis a los cifrados en bloques binarios (Sección 2.3).
2. Los programas elaborados son flexibles, al permitir variaciones en los métodos de selección de parejas, de mutación y cruzamiento, así como en el cifrado a utilizar.
3. Con la función de aptitud introducida en este trabajo se logra un acercamiento mucho mayor, que el obtenido en trabajos anteriores, a la clave real que fue utilizada en la sesión de cifrado.
4. Consideramos que es posible obtener mejores resultados en los experimentos por diferentes vías, por ejemplo: modificando los métodos de selección de parejas y los tipos de cruzamientos y mutaciones, aumentando tanto el tamaño de las poblaciones como la cantidad de generaciones. También se pueden valorar otros posibles pesos en la función de aptitud presentada en este trabajo.
5. Finalmente, pero no menos importante, debe explorarse la utilización del método en cifrados en bloques de mayor fortaleza criptográfica.

Referencias

[1] H. Delfs and H. Knebl. Introduction to Cryptography. Principles and Applications. Springer-Verlag, Second Edition, 2007.

[2] M. Eichlseder, F. Mendel, T. Nad et al. Linear Propagation in Efficient Guess-and-Determine Attacks. International Workshop on Coding and Cryptography, Bergen (Norway). WCC 2013. http://www.selmer.uib.no/WCC2013/pdfs/Eichlseder.pdf.

[3] A. Garrett, J. Hamilton and G. Dozier. A Comparison of Genetic Algorithm Techniques for the Cryptanalysis of TEA. International Journal of Intelligent Control and Systems. Vol. 12, No. 4, 325-330, 2007.

[4] E.D. Goodman. Introduction to Genetic Algorithms. A Tutorial. World Summit on Genetic and Evolutionary Computation. Shanghai, China. 2009. http://www.egr.msu.edu/~goodman/ GECSummitIntroToGA_Tutorial-goodman.pdf.

[5] H.M. Hasan Husei et al. A Genetic Algorithm for Cryptanalysis with Application to DES-like Systems. International Journal of Network Security, Vol. 8, No. 2, pp. 177-186, 2009.

[6] H.M. Heys. A Tutorial on Linear and Differential Cryptanalysis. Cryptologia, 26 (3): 189-221, 2002.

[7] J. Kelsey, B. Schneier, D. Wagner et al. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, V. 8, N. 2-3, pp. 141-158, 2000.

[8] M. Lozano, F. Herrera. SF1: Computación Evolutiva y Algoritmos Bioinspirados. Sesión 1.a: Algoritmos Genéticos y Sistemas Inteligentes de Enjambre. Máster Oficial de la Universidad de Granada. Soft Computing y Sistemas Inteligentes. Curso 2010-2011. http://150.214.190.154/docencia/sf1/SF1-1011-1a.pdf.

[9] G. Patel. Seminar Report on Genetic Algorithm for Cryptanalysis. Birla Institute of Technology & Science (BITS) Pilani, 2008. http://csis.bits-pilani.ac.in/faculty/murali/netsec-09/seminar/ refs/girishprep.pdf.

[10] A. Pérez Serrada. Una Introducción a la Computación Evolutiva. 1996. ftp://ftp.de.uu.net/pub/research/softcomp/EC/EA/ papers/intro-spanish.ps.gz.

[11] R. Vimalathithan and M.L. Valarmathi. Cryptanalysis of DES using Computational Intelligence. European Journal of Scientific Research, 55 (2): 237-244, 2011.