Available online at www.sciencedirect.com
SCIENCE DIRECT®
Nonlinear Analysis 63 (2005) e823–e830
Nonlinear Analysis
www.elsevier.com/locate/na

Evolutionary computation based cryptanalysis: A first study☆

E.C. Laskari[a, b], G.C. Meletiou[c, *], Y.C. Stamatiou[d], M.N. Vrahatis[a, b]

[a] *Computational Intelligence Laboratory, Department of Mathematics, University of Patras, GR–26110 Patras, Greece*
[b] *University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR–26110 Patras, Greece*
[c] *A.T.E.I. of Epirus, P.O. Box 110, GR–47100 Arta, Greece*
[d] *Department of Mathematics, University of the Aegean, GR–83200 Samos, Greece*

Abstract

In this contribution we apply the particle swarm optimization method, which originates from the field of evolutionary computation, to address an interesting problem introduced by the cryptanalysis of block-cipher cryptosystems. The results on the data encryption standard reduced to four rounds indicate that this is a promising approach.

Keywords: Evolutionary computation; Particle swarm optimization; Data encryption standard; Differential cryptanalysis; Feistel cryptosystems

1. Introduction

Evolutionary computation algorithms are stochastic optimization methods that involve algorithmic mechanisms inspired by natural evolution and social behavior. These methods have been proven to be efficient and effective where deterministic optimization methods fail and can handle problems that involve discontinuous objective functions and disjoint search

☆ The authors acknowledge the partial support by the "Archimedes" research programme awarded by the Greek Ministry of Education and Religious Affairs and the European Union.

* Corresponding author.
E-mail address: gmelet@teiep.gr (G.C. Meletiou).

0362-546X/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.na.2005.03.004

spaces [7,10,20]. Commonly encountered paradigms of such methods are genetic algorithms (GA), evolution strategies (ES), differential evolution algorithm (DE) and the particle swarm optimization (PSO). GA and ES are based on the principles of natural evolution. On the other hand, PSO is based on the simulation of social behavior. Optimization techniques for real search spaces can be applied on integer programming problems through slight modification. Usually, the optimum solution is determined by rounding off the real optimum values to the nearest integer [19]. Early approaches in the direction of evolutionary algorithms for integer programming problems are reported in [8,9].

In this paper we study the use of the particle swarm optimization method to address a problem introduced by the cryptanalysis of block-cipher cryptosystems. More specifically, we investigate the problem of finding some missing bits of the key used to a simplified Feistel cipher, the data encryption standard (DES) reduced to four rounds. Our first results are encouraging since the method managed to locate the missing bits on an average of 1500 function evaluations as opposed to the $2^{14} = 16384$ required by brute force. Furthermore, the method can be readily adapted to handle more complex Feistel based ciphers.

The rest of the paper is organized as follows: in Section 2 the basics on block-cipher cryptosystems and differential cryptanalysis are briefly reviewed and the optimization problem is formulated. In Section 3 the considered PSO algorithm is briefly described. In Section 4 the experimental results are reported. In Section 5 conclusions are derived and directions for future work are suggested.

2. Background and problem formulation

In order to describe the problem at hand, let us briefly review the block-cipher cryptosystems and differential cryptanalysis.

2.1. Block-cipher cryptosystems and differential cryptanalysis

An n-bit *block-cipher* is a function $E : V_n \times \mathcal{K} \mapsto V_n$, such that for each key $K \in \mathcal{K}$, and plaintext P, $E(P, K)$ is an invertible mapping called *encryption function*. We denote as $C = E(P, K)$ the ciphertext that results from the encrypting plaintext P under K. The inverse mapping is called *decryption function*.

An *iterated block-cipher* is a block-cipher that is based on sequential r times repetition of a function, the *round function*. Each repetition is called a *round* and the cryptosystem is called an *r-round cryptosystem*. The parameters of the round function include the number of rounds r, the block bitsize n, and the bitsize k of the key K, from which r subkeys K_i (round keys) are derived. The subkeys K_i are calculated via the *key scheduling* algorithm.

The round function is usually based on substitution mappings, called *S-boxes*, bit permutations, arithmetic operations and XOR operations (denoted by $\oplus$). The S-boxes are nonlinear mappings and usually they are the only nonlinear part of the cryptosystem, rendering thus cryptosystem's security crucially depending on them. Due to the importance of the role of substitutions, the engineering issues of S-boxes design and construction have received considerable attention [1,15].

A *Feistel cipher* is an iterated block cipher, mapping an n-bit plaintext P, to a ciphertext C, through an r-round process, where the current n-bit word is divided into $(n/2)$-bit parts, the left part L_i and the right part R_i [6]. Then round i, $1 \leqslant i \leqslant r$ has the following effect:

$$L_i = R_{i-1},$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i),$$

where K_i is the subkey used in the ith round (derived from the cipher key K), and F is an arbitrary round function. The number of rounds is often an even number. The output of a Feistel cipher is ordered as (R_r, L_r) and not as (L_r, R_r) (i.e. after the last round function has been applied, the two halves are swapped).

A nice characteristic of Feistel-based ciphers is that the decrypt function is identical to the encrypt function except that the subkeys and the round functions are applied in reverse. This makes the Feistel structure an attractive choice for both software and hardware implementations.

The best known and most widely used Feistel block-cipher cryptosystem is DES. DES is the outcome of the collaboration between the government of the United States and IBM in the 1970s and today. It is a *symmetric algorithm*, meaning that the parties exchanging information possess the same key. DES processes plaintext blocks of $n = 64$ bits, producing 64-bit ciphertext blocks, with effective key size $k = 64$ bits, 8 of which can be used as parity bits. The plaintext block is divided into the left and right parts of 32 bits each. The main part of the round function is the F function, which works on the right half of the data, using a subkey of 48 bits and eight (6–4 bits) S-boxes. The 32 output bits of the F function are XORed with the left half of the data and the two halves are exchanged. A more detailed description of the DES algorithm can be found in [14,22].

Two of the most powerful cryptanalytic attacks for Feistel-based ciphers, that were first applied with success to the cryptanalysis of DES, depend critically on the exploitation of specific weaknesses of the S-boxes of the target cryptoalgorithm. These attacks are the *Linear Cryptanalysis* (see [13,12]) and the *Differential Cryptanalysis* (see [2,3]).

Differential cryptanalysis (DC) is a chosen plaintext attack which uses only the resultant ciphertexts. The basic tool of the attack is the *ciphertext pair* which is a pair of ciphertexts whose plaintexts have particular differences. The two plaintexts can be chosen at random, as long as they satisfy the difference condition. The method analyzes the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs. These differences can be used to assign probabilities to the possible keys and to locate the most probable key. This method usually works on many pairs of plaintexts with the same particular difference using only the resultant ciphertext pairs. For cryptosystems similar to DES, the difference is chosen as a fixed XORed value of the two plaintexts.

The most important component in DC is the use of a *characteristic*. An informal definition of a characteristic is the following. "Associated with any pair of encryptions are the XOR value of its two plaintexts, the XOR of its ciphertexts, the XORs of the input of each round in the two executions and the XORs of the outputs of each round in the two executions. These XOR values form an *r-round characteristic*. A characteristic has a probability, which is the probability that a random pair with the chosen plaintext XOR has the round and ciphertext XORs specified in the characteristic." [2].

Each characteristic allows looking for a particular number of bits in the subkey of the last round and more specifically for all the bits that enter some particular (but not all) S-boxes. The most useful characteristics are those which have a maximal probability and a maximal number of subkey bits whose occurrences can be counted.

The DC is statistical in nature and can fail in rare instances. A more extended analysis on DC and its results on DES for different numbers of rounds can be found in [2].

2.2. Problem formulation

For the differential cryptanalysis of DES reduced to four rounds, Biham and Shamir [2] use a one-round characteristic with probability 1, and at the first step DC provided 42 bits of the subkey of the last round. In the case where the subkeys are calculated with the DES key scheduling algorithm, the 42 bits given by DC are actual key bits of the 56 key bits and there are 14 key bits still missing. A suggestion for finding these key bits was to try all the 2^{14} possibilities in decrypting the given ciphertexts, using the resulting keys. The right key should satisfy the known plaintext XOR value for all the pairs that are used by DC. The rest $2^{14} - 1$ values of the key have only probability 2^{-64} to satisfy the pairs condition [2].

Instead of using brute force to find the missing key bits, we formulate the problem of the missing 14 bits to an optimization one, as follows. We consider each one of the 14 bits as a component of a 14th dimensional vector. Such a vector represents a possible solution of the problem. Now, assume that the right 42 key bits found by DC were suggested using np pairs. We can use these np pairs to evaluate the possible solutions provided by the optimization method. More specifically, for each possible solution, X_i, suggested by the optimization algorithm, we construct the 56 bits of the key, using the 42 bits which are known by DC and the 14 components of X_i in proper order. With the resulting key, we decrypt the np ciphertext pairs that where used by DC and count the number of decrypted pairs that satisfy the known plaintext XOR value, denoted as cnp_{X_i}. Thus, the evaluation function f, is the difference between the desired output np and the actual output cnp_{X_i}, i.e.,

$$f(X_i) = np - cnp_{X_i}.$$

The global minimum of the function f is zero and the global minimizer provided, will be the actual key with probability $P = 1 - 2^{-64}$.

3. The considered optimization method

For completeness purposes, in this section we briefly describe the considered evolutionary computation method.

PSO is a population-based algorithm that exploits a population of individuals, to search promising regions of the function space. In this context, the population is called *swarm* and the individuals are called *particles*. Each particle moves with an adaptable velocity within the search space, and retains in its memory the best position it ever encountered. In the *global* variant of the PSO the best position ever attained by all individuals of the swarm is communicated to all the particles. In the *local* variant, each particle is assigned to a neighborhood consisting of a prespecified number of particles. In this case, the best

position ever attained by the particles that comprise the neighborhood is communicated among them [5].

Assume a D-dimensional search space, $\mathbf{S} \subset \mathbb{R}^D$, and a swarm of N particles. The ith particle is in effect a D-dimensional vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})^\top$. The velocity of this particle is also a D-dimensional vector, $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})^\top$. The best previous position ever encountered by the ith particle is a point in $\mathbf{S}$, denoted by $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})^\top$. Assume g, to be the index of the particle that attained the best previous position among all the individuals of the swarm. Then, according to the *constriction factor* version of PSO the swarm is manipulated using the following equations [4]:

$$V_i^{(t+1)} = \chi(V_i^{(t)} + c_1 r_1 (P_i^{(t)} - X_i^{(t)}) + c_2 r_2 (P_g^{(t)} - X_i^{(t)})), \tag{1}$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)}, \tag{2}$$

where $i = 1, 2, \ldots, N$; χ is the constriction factor; c_1 and c_2 denote the *cognitive* and *social* parameters, respectively; r_1, r_2 are random numbers uniformly distributed in the range [0, 1]; and t, stands for the counter of iterations. The value of the constriction factor is typically obtained through the formula $\chi = 2\kappa/|2 - \phi - \sqrt{\phi^2 - 4\phi}|$, for $\phi > 4$, where $\phi = c_1 + c_2$, and $\kappa = 1$. Different configurations of χ as well as a theoretical analysis of the derivation of the above formula can be found in [4].

In a different version of PSO a parameter called *inertia weight* is used, and the swarm is manipulated according to the formulae [10,5,21]

$$V_i^{(t+1)} = w V_i^{(t)} + c_1 r_1 (P_i^{(t)} - X_i^{(t)}) + c_2 r_2 (P_g^{(t)} - X_i^{(t)}), \tag{3}$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)}, \tag{4}$$

where $i = 1, 2, \ldots, N$; and w is the inertia weight, while all other variables are the same as in the constriction factor version. There is no explicit formula for the determination of the factor w, which controls the impact of the previous history of velocities on the current one. However, since a large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration (fine-tuning the current search area), it appears intuitively appealing to initially set it to a large value and to gradually decrease it to obtain more refined solutions. The superiority of this approach against the selection of a constant inertia weight, has been experimentally verified [21]. Thus, an initial value around 1.2 and a gradual decline toward 0.1 can be considered as a good choice for w. Proper fine-tuning of the parameters c_1 and c_2, results in faster convergence and alleviation of local minima. As default values, $c_1 = c_2 = 2$ have been proposed, but experimental results indicate that alternative configurations, depending on the problem at hand, can produce superior performance [10,16].

Typically, the swarm and the velocities, are initialized randomly in the search space. For more sophisticated techniques, see [17]. For uniform random initialization in a multidimensional search space, a Sobol sequence generator can be used [18]. The performance of the PSO method for the integer programming problem was studied in [11] with very promising results.

4. Experimental setup and results

PSO method was applied considering each component of the possible solution as a real number in the range [0, 1]. For the evaluation of the suggested solutions PSO was applied by rounding off real values to the nearest integer. As the constriction factor version of PSO is considered faster at converging to a solution, the global and local PSO variants of this version of the method, were tested. All populations were constrained in the feasible region of the problem and the size of each population was taken equal to 100. The maximum velocity, $V_{\max}$, of the method was set to 0.5. The parameters of the method were set at the default values, i.e. $\chi = 0.729$ and $c_1 = c_2 = 2.05$, found in the literature [4]. The proposed approach was tested for several different initial keys and number of pairs, np. For each setting, the performance of the method was investigated on 100 independent runs. The results for six different keys, k_i, $i = 1, \ldots, 6$, and for test pairs, np, equal to 20, 50 and 100, are reported in Tables 1–3, respectively.

Concerning the notation used in the tables, PSOCG is the global variant of PSO with constriction factor and PSOCL is PSO's local variant with constriction factor. A run is considered to be successful if the algorithm identifies the global minimizer within a prespecified number of function evaluations. The function evaluations threshold was taken equal to 2^{14}. The success rates of each algorithm, that is the proportion of the times it achieved the global minimizer within the prespecified threshold, the minimum number and the mean value of function evaluations used by the method, are reported.

Our first results are encouraging since the method managed to locate the missing bits on an average of 1500 function evaluations as opposed to the $2^{14} = 16,384$ required by brute force. Relative to the different variants of the PSO method, the local variant accomplished higher success rates within the prespecified threshold of function evaluations. Finally, using a larger number, np, of test pairs speeds up the convergence rate, as the function value of local minima seems to be elevated.

Table 1
Results for six different keys using $np = 20$ test pairs

Key	Method	Suc.rate	Function evaluations	
		(%)	Mean	Min
k_1	PSOCG	98	1146	200
k_1	PSOCL	100	2020	200
k_2	PSOCG	99	854	200
k_2	PSOCL	100	2079	200
k_3	PSOCG	97	1542	200
k_3	PSOCL	100	2300	200
k_4	PSOCG	97	1698	200
k_4	PSOCL	100	1884	300
k_5	PSOCG	93	1870	200
k_5	PSOCL	100	1788	300
k_6	PSOCG	100	740	200
k_6	PSOCL	100	1717	200

Table 2
Results for six different keys using $np = 50$ test pairs

Key	Method	Suc.rate	Function evaluations	
		(%)	Mean	Min
k_1	PSOCG	99	885	200
k_1	PSOCL	100	1873	200
k_2	PSOCG	100	682	200
k_2	PSOCL	100	1348	200
k_3	PSOCG	100	606	200
k_3	PSOCL	100	1432	200
k_4	PSOCG	96	1322	200
k_4	PSOCL	100	1382	200
k_5	PSOCG	98	941	200
k_5	PSOCL	100	1691	200
k_6	PSOCG	96	1205	200
k_6	PSOCL	100	1627	200

Table 3
Results for six different keys using $np = 100$ test pairs

Key	Method	Suc.rate	Function evaluations	
		(%)	Mean	Min
k_1	PSOCG	100	640	200
k_1	PSOCL	100	1225	200
k_2	PSOCG	97	1082	200
k_2	PSOCL	100	1261	200
k_3	PSOCG	99	833	300
k_3	PSOCL	100	1633	200
k_4	PSOCG	100	589	200
k_4	PSOCL	100	1255	200
k_5	PSOCG	99	883	200
k_5	PSOCL	100	1214	200
k_6	PSOCG	92	2043	200
k_6	PSOCL	100	1640	200

5. Conclusions and future work

The particle swarm optimization method was applied to the problem of locating the key of a simplified version of DES [2]. As reported in [2], in the case of keys produced by the DES key scheduling algorithm, the determination of the key bits involves a brute force search for 14 missing bits of a previously computed part of the key. The PSO method, as evidenced by the experimental results given in the previous section, seems to speed-up considerably the search for the 14 missing bits. This suggests that the two methods can be used complementary to each other, in order to locate all the bits of the key.

We also observed that when the input and output permutation functions of DES are omitted, PSO actually locates 4 candidate values for the 14 missing bits that minimize the evaluation function, which differ in 2 fixed positions that correspond to positions 10 and 36 of the DES key. However, the permutations do not affect the method, but the ciphertext pairs which are imported to the evaluation function.

In conclusion, in view of the encouraging results of our preliminary experiments, we believe that the PSO method will also be beneficial in locating bits of the key other than the 14 bits on which we applied it.

References

[1] C. Adams, Constructing symmetric ciphers using the cast design procedure, in: Proceedings on Design, Codes, and Cryptography, 1997, pp. 283–316.
[2] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems, J. Cryptol. (1991).
[3] E. Biham, A. Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer, Berlin, 1993.
[4] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (1) (2002) 58–73.
[5] R.C. Eberhart, P. Simpson, R. Dobbins, Computational Intelligence PC Tools, Academic Press, New York, 1996.
[6] H. Feistel, Cryptography and computer privacy, Sci. Am. (1973).
[7] D.B. Fogel, Evolutionary Computation: towards a New Philosophy of Machine Intelligence, IEEE Press, Piscataway, NJ, 1995.
[8] D.A. Gall, A practical multifactor optimization criterion, in: T.P. Vogl (Ed.), Recent Advances in Optimization Techniques, 1966, pp. 369–386.
[9] R.C. Kelahan, J.L. Gaddy, Application of the adaptive random search to discrete and mixed integer optimization, Int. J. Numer. Meth. Eng. 12 (1978) 289–298.
[10] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufmann, Los Altos, CA, 2001.
[11] E.C. Laskari, K.E. Parsopoulos, M.N. Vrahatis, Particle swarm optimization for integer programming, in: Proceedings of the IEEE 2002 Congress on Evolutionary Computation, Hawaii, HI, 2002, IEEE Press, New York, pp. 1576–1581.
[12] M. Matsui, Linear cryptanalysis method for DES cipher, Lect. Notes Comput. Sci. 765 (1994) 386–397.
[13] M. Matsui, A. Yamagishi, A new method for known plaintext attack of feal cipher, Lect. Notes Comput. Sci. (1992), 81–91.
[14] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography CRC Press Series on Discrete Mathematics and its Applications, CRC Press, Boca Raton, FL, 1996.
[15] S. Mister, C. Adams, Practical S-box design, in: Proceedings of the Third Annual Workshop on Selected Areas in Cryptography, 1996.
[16] K.E. Parsopoulos, M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Nat. Comput. 1 (2002) 235–306.
[17] K.E. Parsopoulos, M.N. Vrahatis, Initializing the particle swarm optimizer using the nonlinear simplex method, in: A. Grmela, N.E. Mastorakis (Eds.), Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS Press, 2002, pp. 216–221.
[18] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in Fortran 77, Cambridge University Press, Cambridge, 1992.
[19] S.S. Rao, Engineering optimization—Theory and Practice, Wiley Eastern, New Delhi, 1996.
[20] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.
[21] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE Conference on Evolutionary Computation, Anchorage, AK, 1998.
[22] D. Stinson, Cryptography: Theory and Practice (Discrete Mathematics and its Applications), CRC Press, Boca Raton, FL, 1995.