Cryptanalysis of SDES Using Modified Version of Binary Particle Swarm Optimization

Kamil Dworak[1,2(✉)] and Urszula Boryczka[2]

[1] Future Processing, Bojkowska 37A, 44-100 Gliwice, Poland
[2] Institute of Computer Science, University of Silesia,
Bedzinska 39, 41-200 Sosnowiec, Poland
kamil.dworak@us.edu.pl, urszula.boryczka@us.edu.pl
http://www.ii.us.edu.pl/en
http://www.future-processing.com

Abstract. Nowadays, information security is based on ciphers and cryptographic systems. What evaluates the quality of such security measures is cryptanalysis. This paper presents a new cryptanalysis attack aimed at a ciphertext generated with the use of the *SDES* (Simplified Data Encryption Standard). The attack was carried out with a modified version of the *BPSO* (Binary Particle Swarm Optimization) algorithm. A well-adjusted version of this method can have a positive effect on the quality of the results obtained in a given period of time.

Keywords: Simplified Data Encryption Standard · Binary Particle Swarm Optimization · Particle Swarm Optimization · Cryptanalysis · Cryptography

1 Introduction

Security based on authentication is not sufficient these days. Intercepting any sensitive data may happen even during the process of developing computer systems. In these cases, cryptography is becoming more and more popular. In particular, this field deals with ciphering and deciphering information, but it also designs complex cryptographic systems assuring safe data access [6]. Its main objective is not to obscure the existence of the message, but to transform it in such a way that it can only be read by its sender and intended recipients [12]. Cryptography is closely related to cryptanalysis, a field of study which is about deciphering ciphertexts without knowing the right deciphering key [5]. It is also used in the process of finding bugs in existing cryptographic systems to increase and upgrade their level of security [1]. Cryptanalysis processes are not the fastest ones, their operation time is very long. In order to optimize the time of these operations, one of the optimization techniques called *BPSO* was used.

Techniques based on artificial intelligence are more commonly used in the field of computer security. Over recent years, many publications on the application of various evolutionary techniques such as Tabu Search, evolutionary algorithms or Simulated Annealing in cryptanalysis, have been presented. In 1998

M. Núñez et al. (Eds.): ICCCI 2015, Part II, LNCS 9330, pp. 159–168, 2015.
DOI: 10.1007/978-3-319-24306-1_16

Andrew John Clark [2] presented the results of his research on the cryptanalysis of classic ciphers with the use of evolutionary algorithms. The results were not accurate every time, though in most cases it was possible to read the decrypted message. Poonan Garg in [4] presented an interesting attack carried out using a memetic algorithm on a ciphertext generated with the *SDES*, which had very good results. Nalini and Raghavendra presented in their article [10] a cryptanalysis attack using *PSO* with grouping against the modification of *DES*, developed by the authors themselves. Techniques based on evolutionary algorithms are gaining more interest in the field of computer security. It leads to many successes, yet there are still numerous problems to be discussed [1]. The next chapter of this paper presents the specification of the *SDES* used to generate a ciphertext that will be subjected to the attack. The third chapter contains the basic information on *PSO* and *BPSO* algorithms and their modifications for the purpose of cryptanalysis. The next chapter concerns the experiments, observations and the results of decrypting the ciphertext using the developed attack and comparison between proposed attack and the simple random walk algorithm. The last chapter contains conclusions and further plans.

2 Simplified Data Encryption Standard

The *SDES* is a simplified version of a well-known Data Encryption Standard (*DES*). It was designed in 1996 by Edward Schaefer for general academic purposes [11]. *SDES* is a symmetric cipher, that is, the same key is used to encrypt and decrypt information [12]. It is a block cipher. The length of a single encrypted message block equals 8 bits and the data is encrypted by a 10-bit key [11]. The *SDES* operates on strings of bits instead of normal characters. The decryption process uses the same encrypting algorithm but the subkeys are provided in reverse order. *SDES* is a two-round algorithm, based on two basic operations: combination done with permutations and dispersion [9]. The block diagram of the encryption is shown in Fig. 1.

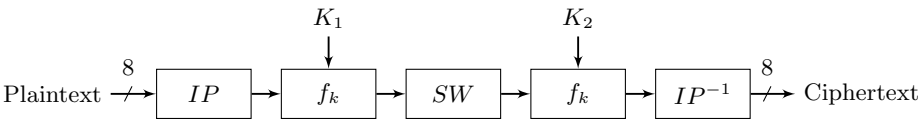

Fig. 1. Encryption with the use of *SDES*

The first round is preceeded by the initial permutation IP [11]. The bits of a message are relocated with the use of an appropriate, determined shift table. The second bit is put to the first position, the sixth one to the second etc. A moved string is then divided into two equal 4-bit parts (L - left and R - right), and the f_k, function, known as the round function, is executed:

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

where:

f_{K_i} - round function,
L/R - left/right part of the binary (input) string,
K_i - round key,
f - deciphering function,
i - round's ID.

The process of swapping the right part to the left is executed by the SW function (swap). Then the second round begins. This time the SW function is omitted. At the end, the result is put through the final IP^{-1} permutation, which is the opposite of the initial permutation IP [9]. In this way, an 8-bit block of ciphertext is generated. The operation is repeated until the plaintext is encrypted. The register of all essential permutations looks as follows:

$$\begin{array}{lll}
IP & = [& 2\ 6\ 3\ 1\ 4\ 8\ 5\ 7\ ] \\
P10 & = [& 3\ 5\ 2\ 7\ 4\ 10\ 1\ 9\ 8\ 6\ ] \\
P8 & = [& 6\ 3\ 7\ 4\ 8\ 5\ 10\ 9\ ] \\
E & = [& 4\ 1\ 2\ 3\ 2\ 3\ 4\ 1\ ] \\
P4 & = [& 2\ 4\ 3\ 1\ ] \\
IP^{-1} & = [& 4\ 1\ 3\ 5\ 7\ 2\ 8\ 6\ ]
\end{array}$$

2.1 Generating Subkeys

In order to properly calculate the value of the f_k function, it is necessary to generate the subkeys. For every round, the key is modified with such operations as reduction, shifting and permutation [11], the aim of which is to generate an auxiliary key. At the beginning, the encryption key is subjected to the given permutation *P10*. A received string is divided into two 5-bits parts. A single cyclical shift to the left is performed on each of them. The shifted parts are concatenated in a binary string. The final string is subjected to the *P8*, permutation, thereby generating the K_1 subkey. The K_2 subkey is the result of an auxiliary, double, cyclical shift at the stage of dividing the encryption key. A received string is also compressed with the *P8* permutation.

2.2 f_{K_i} and f Functions

The f_{K_i} function is executed by the *Feistal network* (Fig. 2). The right part of the string is automatically moved to the left side. The left part is subjected to the addition modulo 2 with the result of the ciphering function f, the arguments of which are the K_i subkey and the right part of the string of R_{i-1}.

The function f is the right ciphering algorithm. A flow chart illustrating its performance is shown in (Fig. 3). At the beginning, the expanding permutation E is executed. To increase the coefficient of the avalanche effect, two new settings

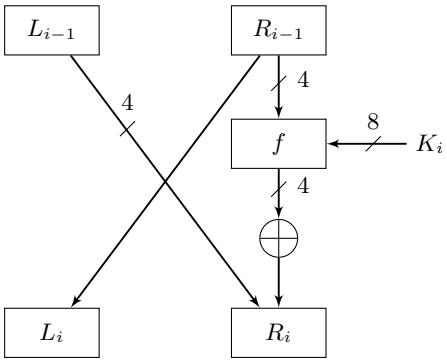

Fig. 2. Feistal function f_{K_i}

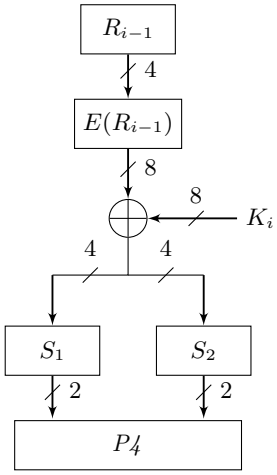

Fig. 3. Deciphering function f

from the determined permutation E [12] are assigned to every bit. The right part of the string is expanded from 4 to 8 bits. In this way, it is possible to perform the additional modulo 2 with the given K_i subkey. The received string is divided into two equal parts. The left one is moved to the *S-Block* S_1, whereas the right one to S_2. Each block is a specially designed 4x4 matrix of the elements from the set 0, 1, 2, 3 written in the binary system [11]. The *S-Blocks* are the only non-linear element of majority of block ciphers [12,14]. The 4-bit strings of data are the input. They are converted into 2-bit output. The first and last bit of each substring represents the number of a row, whereas the second and third bit represents the number of a column. The *S-Blocks* are presented below. The two chosen digits are concatenated with each other. The result is finally subjected to the *P4* permutation.

$$S_1 = \begin{bmatrix} 01 & 00 & 11 & 10 \\ 11 & 10 & 01 & 00 \\ 00 & 10 & 01 & 11 \\ 11 & 01 & 11 & 10 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 00 & 01 & 10 & 11 \\ 10 & 00 & 01 & 11 \\ 11 & 00 & 01 & 00 \\ 10 & 01 & 00 & 11 \end{bmatrix}.$$

3 Particle Swarm Optimization

The *PSO* was first presented in 1995 by Kennedy and Eberhart [7]. The authors were inspired by the behavior of animals in troops. They attempted to simulate the behavior of a school of fish moving in a precise and specific manner. In case of danger, the behavior of an individual has a dramatic impact on the behavior of the whole population. Another example of their inspiration is a bevy of birds searching for food. When one of the birds finds a good source of food, the whole bevy moves in this direction. Similarly to evolutionary algorithms, a single

individual is represented by a *particle*, whereas the population is a swarm [3]. The particles are dispersed within a multidimensional space of solutions in order to find the global extreme. They move according to their own experience or the one of other, neighboring particles [3]. Every individual is characterized by the speed vector $\overrightarrow{V}$ and the position vector $\overrightarrow{X}$ [7]. Initially, the particles are randomly placed in the space. In each iteration, the value of the adjustment function is calculated for every particle. During this operation, so far the best adjustment of a given P_{BEST} particle and so far the best adjustment among all of the particles of the swarm G_{BEST} are determined. Then, for every individual, a new value of the speed vector (the below formula) is calculated and the particle is moved to a new position:

$$\overrightarrow{V}(t+1) = \overrightarrow{V} + c_1 r_1 \cdot (\overrightarrow{X}_{P_{BEST}} - \overrightarrow{X}) + c_2 r_2 \cdot (\overrightarrow{X}_{G_{BEST}} - \overrightarrow{X}), \tag{2}$$

where:
$\overrightarrow{V}$ - current speed vector,
c_1, c_2 - positive acceleration constants,
r_1, r_2 - random numbers from [0, 1],
$\overrightarrow{X}$ - current particle position vector,
$\overrightarrow{X}_{P_{BEST}}$ - best position of the particle so far,
$\overrightarrow{X}_{G_{BEST}}$ - best position among all the particles so far,
t - iteration of the algorithm.

In the case where the speed value $\overrightarrow{V}(t+1)$ is greater than the established acceptable particle speed V_{MAX}, the following formula applies:

$$\overrightarrow{V}(t+1) = V_{MAX}. \tag{3}$$

3.1 Binary Particle Swarm Optimization

In 1997 Kennedy and Eberhart developed a modified version of the *PSO*, dedicated to problems characterized by a binary space of solutions, i.e. *BPSO* [8]. Every element of the vector is represented by binary values, namely $X_{i,j} \in \{0, 1\}$. Changing the position of a particle is reduced to negation of bits of particular indexes [3]. One of the most important changes concerning *BPSO* is an update of the $\overrightarrow{X}$ position vector, which is determined with the formula [3]:

$$\overrightarrow{X}_{i,j}(t+1) = \begin{cases} 1 & if \quad r_{3_j}(t) < sig(\overrightarrow{V}_{i,j}(t+1)), \\ 0 & otherwise \end{cases} \tag{4}$$

where:
r_3 - random numbers from [0, 1],
sig - sigmoidal function,
i - particle index,
j - position in the speed / position vector,
t - iteration of the algorithm.

The sigmoidal function value can be calculated using the following formula [3]:

$$sig(\overrightarrow{V}_{i,j}(t)) = \frac{1}{1 + e^{-\overrightarrow{V}_{i,j}(t)}} \quad , \quad sig(\overrightarrow{V}) \in [0, 1]. \tag{5}$$

3.2 Proposed Attack

The suggested algorithm is based on the chosen-plaintext attack, which is one of the most popular cryptanalytical methods. This attack is referred to as "Particle Swarm Attack". The attacker is in the possession of the ciphertext enciphered with the *SDES* and the corresponding plaintext [5,6]. Using these data, the cryptanalyst tries to guess what the deciphering key is, the aim of which is to decipher the rest of the intercepted ciphertexts. The algorithm begins with generating a defined number of particles with the speed equal to 0. The $\overrightarrow{X}$ position vector is equated with the example of a decryption key. In the initial iteration the keys are generated randomly. Next, based on the Hamming distance and according to the formula below, the value of the adjustment function is calculated:

$$F_f = H(P, D) = \sum_{i=1}^{n} P_i \oplus D_i, \tag{6}$$

where:

D_i - the character of the deciphered text,
P_i - the character of the plaintext,
H - Hamming distance,
n - message length.

Then, the P_{BEST} and G_{BEST} values are determined. To do so, the modification to the classic version of *PSO* [13] will be applied. Having calculated the adjustment value for every particle, an auxiliary steering parameter called statistical probability α is introduced. In most cases, it is represented by a random number from [13]. In the proposed attack it will have one constant value. Then, based on the α and speed vector $\overrightarrow{V}$, the particle can update the chosen fragments of the $\overrightarrow{X}$ position vector in relation to its best so far position P_{BEST} or the best position in the G_{BEST} swarm. It is done according to the following formula:

$$if \quad (\alpha < \overrightarrow{V}_i \leq (\frac{1}{2}(1 + \alpha)), \quad then \overrightarrow{X}_i = P_{BEST_i}, \tag{7}$$

$$if \quad (\frac{1}{2}(1 + \alpha) < \overrightarrow{V}_i \leq 1), \quad then \overrightarrow{X}_i = G_{BESTi}. \tag{8}$$

Otherwise, when $(0 < \overrightarrow{V}_i \leq \alpha)$, the original version of the modification presented in [13], considers the application of the previous position vector.

In the developed attack, it is suggested to calculate new value of the vector according to the classic *BPSO* algorithm:

$$\overrightarrow{V}_{i+1}(t+1) = \overrightarrow{V}_i + c_1 r_1 \cdot (\overrightarrow{X}_{P_{BEST_i}} - \overrightarrow{X_i}) + c_2 r_2 \cdot (\overrightarrow{X}_{G_{BEST_i}} - \overrightarrow{X_i}), \quad (9)$$

where:

i - particle's index,
j - position in the speed/position vector.

A new value of the position vector is calculated using the eq. 4.

4 Experiments

The aim of the research was to prove that the algorithm presented in this article is able to find the correct ciphering key for a given ciphertext. The attack was implemented in C++ and tested on a computer with an Intel i7 processor clocked at 2.1 GHz. All the tests were performed on one core. The tested ciphertexts were of various length and were created with randomly generated cipher keys. The plain texts were written in English. The algorithm is based on the chosen-plaintext attack. The suggested approach assumes that the ciphering key will be found without searching every possible combination.

All of the control parameters were extracted in an empirical way. The α coefficient is a floating-point number selected randomly. The c_1 cognitive coefficient was set at 0.5, the social factor c_2 at 0.8. The V_{MAX} particle speed was adjusted to the sigmoidal function, to the value 4. The swarm consisted of 20 particles. For every case, the algorithm run 30 times. The experiments were divided according to the length of the ciphertexts into 2 rounds (250 and 1000 characters). Maximum number of iterations was set to 30. *SDES* has only 1024 possible keys so larger amount of iterations is not required.

Table 1 presents the results: the minimum, maximum, median, average and standard deviation of the values of the adjustment functions of the consecutive tests, carried out for Particle Swarm and Single Walk algorithms, for 250 character ciphertexts. In addition, information about the sought after and the best decryption key found are included.

In the first round, 18 out of 30 ciphertexts were cracked by Particle Swarm attack. The Random Walk algorithm cracked only 8 ciphertexts. We decided to compare the guessed keys with the correct ones. In most cases the difference concerned two bits. Fig. 4 shows a boxplot for a few 250-character ciphertexts for Particle Swarm attack. We can see the minimum, maximum and median values for given iterations of the algorithm. The solution is shown in the chart as a unique value.

Table 2 juxtaposes the results for the 1000 character ciphertexts. In this case 24 ciphertexts were properly deciphered by Particle Swarm oppose to the 10 by Random Walk algorithm. Similarly to the previous tests, incorrect decryptions differed by two bits. Fig. 5 shows a box chart for a few 1000-character ciphertexts for Particle Swarm attack.

Table 1. F_f values for each consecutive 250 character ciphertexts for both attacking algorithms

ID	Minimum	Median	Maximum	Average	Standard Deviation	Correct Key	Obtained Key	Iteration
Particle Swarm Attack								
1	0	984	1140	922	251	1000011000	1000011000	17
2	0	1021	1325	972	270	0110101101	0110101101	3
3	0	957	1388	931	290	1010000100	1010000100	12
4	526	940	1202	931	191	000[0]01[1]010	000[1]01[0]010	-
5	0	952	1258	916	258	0101001100	0101001100	10
6	494	1054	1311	1006	200	00[0]100[01]00	00[1]100[10]00	-
7	666	904	1266	945	175	110[1]00[0]100	110[0]00[1]100	-
8	0	1032	1831	983	320	1000001101	1000001101	7
9	526	738	1134	785	186	110[1]00[0]011	110[0]00[1]011	-
10	0	1032	1831	983	320	1011010101	1011010101	10
Random Walk Algorithm								
1	638	939	1321	965	31525	10[0]0[0]11000	10[1]0[1]11000	-
2	731	995	1278	1010	21908	011[0]101101	011[1]101101	-
3	769	1010	1313	1013	22889	101[0]00[0]100	101[1]00[1]100	-
4	517	1039	1149	987	22151	000[0]01[1]010	000[1]01[0]010	-
5	0	918	1161	895	50530	0101001100	0101001100	27
6	590	999	1280	979	30384	000[1]00[0]100	000[0]00[1]100	-
7	727	987	1240	978	19897	1[0]0000110[1]	1[1]0000110[0]	-
8	0	984	1324	933	86186	1000001101	1000001101	12
9	646	982	1210	982	20067	110[1101]011	110[0010]011	-
10	0	983	1172	938	62002	1011010101	1011010101	23

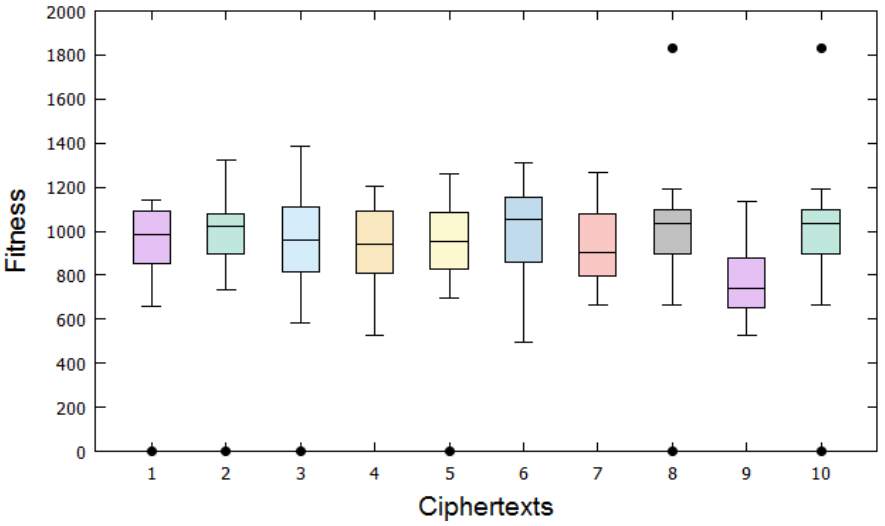

Fig. 4. The boxplot of a randomly picked 250 character ciphertexts for Particle Swarm Attack

Table 2. F_f values for each consecutive 1000 character ciphertexts for both attacking algorithms

ID	Minimum	Median	Maximum	Average	Standard Deviation	Correct Key	Obtained Key	Iteration
Particle Swarm Attack								
11	0	4416	5521	3982	1436	0001101101	0001101101	5
12	0	4063	5336	3993	1077	0000111001	0000111001	7
13	347	3240	4541	2772	1456	001[0]10[0]111	001[1]10[1]111	-
14	0	3811	5340	3614	1453	1000111100	1000111100	17
15	642	4188	5540	3875	1170	100[0]10[0]010	100[1]10[1]010	-
16	0	4554	5522	4062	1404	1010001011	1010001011	12
17	0	4390	5535	4061	1256	1110101000	1110101000	10
18	0	4285	5745	3966	1252	0000110101	0000110101	9
19	0	4103	5086	3777	1106	0001111111	0001111111	5
20	0	4129	4992	3842	1078	1111010000	1111010000	11
Random Walk Algorithm								
11	3007	3845	4816	3874	332190	000[1]101101	000[0]101101	-
12	2744	3558	4414	3566	239546	000[0]111001	000[1]111001	-
13	0	3982	4780	3470	1532160	0010100111	0010100111	23
14	2908	4000	5162	4006	372737	100011[1]100	100011[0]100	-
15	2942	4021	4899	4057	203871	[10]0[0]10[0]0[1]0	[01]0[1]10[1]0[0]0	-
16	0	4105	4971	3769	1245140	1010001011	1010001011	21
17	3184	4193	5073	4179	288314	111[0]10[1]000	111[1]10[0]000	-
18	0	3994	5531	3591	1974630	0000110101	0000110101	16
19	1804	3851	5616	4004	685290	000111[1]111	000111[0]111	-
20	2382	4067	4841	3858	419308	111[1]010000	111[0]010000	-

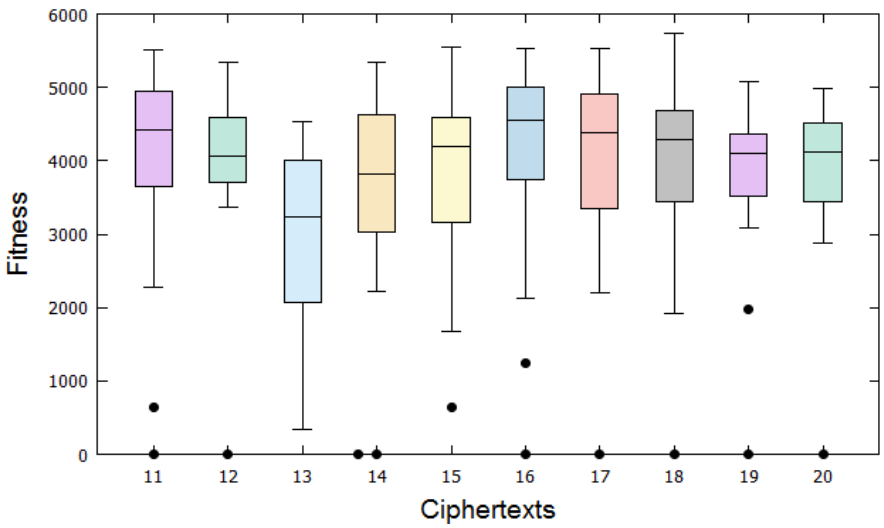

Fig. 5. The boxplot of a randomly picked 1000 character ciphertexts for Particle Swarm Attack

5 Summary

The suggested Particle Swarm cryptanalysis attack allows to obtain satisfying results in the *SDES* cipher cryptanalysis. Correct deciphering can be achieved on the level of up to 80%, depending on the length of the ciphertext.

In the situation when the cryptanalysis attack failed, an interesting correlation between a correct key and the best one found was observed. Mostly, the non-matching bits were symmetrical to each other, as in the test 11. Undoubtedly, it sets out the course for further research in terms of applying modifications in order to improve the quality of obtained results. Moreover, this method should be tested for other ciphering algorithms like *DES* or *Blowfish*. It should be remembered that *BPSO* is one of the possibilities. Another interesting alternative is the Discrete *PSO*, an algorithm, which can perform manage quite well with such problems.

References

1. Boryczka, U., Dworak, K.: Genetic transformation techniques in cryptanalysis. In: Nguyen, N.T., Attachoo, B., Trawiński, B., Somboonviwat, K. (eds.) ACIIDS 2014, Part II. LNCS, vol. 8398, pp. 147–156. Springer, Heidelberg (2014)
2. Clark, A.J.: Optimisation Heuristics for Cryptology. PhD thesis. Information Research Centre Faculty of Information Technology Queensland (1998)
3. Engelbrecht, A.P.: Fundamentals of computional swarm intelligence. Wiley (2005)
4. Garg, P.: Cryptanalysis of SDES via Evolutionary Computation Techniques. International Journal of Computer Science and Information Security **1**(1) (2009)
5. Kahn, D.: The Code-breakers. Scribner (1996)
6. Kenan, K.: Cryptography in the databases. The Last Line of Defense. Addison Wesley Publishing Company (2005)
7. Kennedy, J., Eberhart, R.C.: New optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science (1995)
8. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm optimization. In: Proc. of the Conference on Systems, Man, and Cybernetics SMC 1997 (1997)
9. Menezes, A.J., Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
10. Nalini, N., Raghavendra, R.: Cryptanalysis of Block Ciphers via Improvised Particle Swarm Optimization and Extended Simulated Annealing Techniques. International Journal of Network Security **6**(3) (2008)
11. Schaefer, E.: A simplified data encryption standard algorithm. Cryptologia **20**(1) (1996)
12. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. Wiley (1996)
13. Shen, Q., Jiang, J.H., Jiao, C.X., Shen, G.L., Yu, R.Q.: Modified particle swarm optimization algorithm for variable selection in MLR and PLS modeling: QSAR studies of antagonism of angiotensin II antagonists. European Journal of Pharmaceutical Sciences **22** (2004)
14. Stinson, D.S.: Cryptography. Theory and Practice. Chapman & Hall/CRC Taylor & Francois Group, Boca Raton (2006)