Cryptanalysis of Simplified-DES using Computational Intelligence

VIMALATHITHAN.R[1], M.L.VALARMATHI[2]
[1]Department of ECE, Sri Krishna College of Engineering and Technology
[2]Department of CSE, Government College of Technology
Coimbatore
INDIA
athivimal@gmail.com[1], drmlv@gct.ac.in[2]

Abstract: - Cryptanalysis with Computational Intelligence has gained much interest in recent years. This paper presents an approach for breaking the key used in Simplified-Data Encryption Standard (S-DES) using Genetic algorithm (GA), Particle Swarm Optimization (PSO) and a novel approach called Genetic Swarm Optimization (GSO) obtained by combining the effectiveness of GA and PSO. Ciphertext-only attack is embraced here and an optimum key is produced based on Letter Frequency analysis as Cost function. The key is optimized using the capabilities of Computational Intelligence and the experimental results indicate GSO is an effective tool which runs through less time to break the key used in S-DES and reduces the search space nearly by a factor of 6.

Key-Words: - Cryptanalysis, ciphertext-only attack, Genetic Algorithm, Particle Swarm Optimization , Genetic Swarm Optimization , cost, plaintext and ciphertext.

1. Introduction

Cryptography is the transformation (encryption) of a given message into another message which appears meaningful only to the intended recipient through the process of decryption. The message that undergoes encryption is called the plaintext and the transformed message is called ciphertext. A cryptographic algorithm is a mathematical function employed for the encryption and decryption of messages. Cryptanalysis refers to the process of discovering the plaintext from the ciphertext without knowing the decryption key. Cryptography is the art of making cipher text while Cryptanalysis is the art of breaking ciphertext [1].

Cryptanalysis is a challenging task in Cryptology. There are several types of attacks that a cryptanalyser may use to break a cipher, depending upon how much information is available to the attacker. The goal is to derive the key so that the ciphertext can be easily recovered. An attack on cipher text may be of various types. One type of attack is ciphertext-only attack which is a baffling problem in attacking ciphers and considered in this paper. In this type of attack, the encryption algorithm used and the cipher text to be decoded are known to cryptanalyst. A Brute-force attack is used for ciphertext-only attack where the cryptanalyser tries every possible combination of key until the correct key is identified [2, 3]. The key search space is large for lengthy keys but using a network of computers and combining their computational strength and their cumulative power, Brute-force attack is feasible at increased cost. Instead, using Computational intelligence (CI) the problem can be solved without searching the entire key space. CI has been successfully applied in numerous scientific fields [4, 5]. Here we have applied in the field of cryptanalysis and successful.

CI can be considered as the study of adaptive mechanisms that enable intelligent behaviour of a system in complex and changing environments like Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) [4]. GA and PSO is a population based optimization which could be applied to solve optimization problems. Unlike GA, PSO has no evolution operations like crossover and mutation. The strength of PSO is its fast convergence, which compares favourably with global optimization algorithm like GA. Both GA and PSO share common elements and initialize a population in a similar manner and evaluate a cost function. At last both are generational. By combining the effectiveness of GA and PSO, a new hybrid evolutionary technique called Genetic Swarm Optimization (GSO) is used here, which strongly integrates the vantage characteristics of GA and PSO. The hybrid GSO algorithm is developed in order to overcome the problem of premature convergence.

Several solutions have been proposed in this area. In 1993, for the first time, the paper by Spillman presented a genetic algorithm based

approach for the cryptanalysis of substitution cipher [6]. This paper has explored the possibility of random type search to discover the key (or key space) for a simple substitution cipher.

In 2009, Garg breaks S-DES via Evolutionary Computation, where the Memetic algorithm, GA and Simulated Annealing were used and the results are compared [7]. In the same year, Garg [8] explored the use of Memetic algorithm to break a simplified data encryption standard algorithm and compared the result with GA.

In 2007 Nalini used efficient heuristics to attack S-DES and Modified DES [9]. In 2006, Nalini [10] used optimization heuristics such as GA, Tabu search and Simulated Annealing techniques to break S-DES and compared their performance.

In 2009 Vimalathithan used GA to attack S-DES, by properly tuning the GA parameters; the attack was successful [11]. In 2010, Vimalathithan used PSO to attack S-DES and the result shows PSO performs better than GA [12].

In this paper we propose a novel approach called GSO (by combining the effectiveness of GA and PSO) to break the key used in S-DES using ciphertext-only attack, since it is the most difficult attack among the classes of attacks encountered in cryptanalysis and hence we consider this type of attack. Though S-DES is a much simplified version of DES, cryptanalysis of S-DES will give a better insight into the attack of DES and other block ciphers.

The rest of the paper is organized as follows: In Section 2 we present a brief overview of S-DES and background of CI. In section 3 we describe the Cost function and how CI is used to attack S-DES. Experimental results are presented in Section 4. Finally, section 5 concludes our paper.

2. Overview of Basics

In this section we briefly discuss the basics of S-DES and Computational intelligence.

2.1 Basics of S-DES Algorithm

This section briefly gives the overview of S-DES Algorithm. The SDES encryption algorithm takes an 8-bit block of plaintext and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used for encryption as input and produces the original 8-bit block of plaintext as output. The key generation algorithm and Encryption/Decryption algorithms were discussed below.

2.1.1 Key Generation

For key generation, a 10-bit key is considered from which two 8-bit sub keys are generated. In this case, the key is first subjected to a permutation P10= [3 5 2 7 4 10 1 98 6], then a shift operation is performed. The numbers in the array represent the value of that bit in the original 10-bit key. The output of the shift operation then passes through a permutation function that produces an 8-bit output P8 = [6 3 74 8 5 10 9] for the first sub key (K1). The output of the shift operation also feeds into another shift operation and another instance of Permutation P8 to produce the second sub key K2. In all bit strings, the leftmost position corresponds to the first bit. The key generation algorithm is shown in Figure 1.

2.1.2 Encryption Algorithm

The block schematic of the S-DES encryption algorithm is shown in Figure 1.

The Encryption process involves the sequential application of five functions:

1. Initial and final permutation (IP):

The input to the algorithm is an 8-bit block of plaintext, which is first permuted using the Initial Permutation function IP = [2 6 3 1 4 8 5 7]. Here the bits are mixed according to the IP. At the end of the algorithm, the inverse permutation is applied; the inverse permutation is done by applying,

IP^{-1} = [4 1 3 5 7 2 8 6]

2. Function f_K:

The function f_k, which is the complex component of S-DES, consists of a combination of permutation and substitution functions. The functions are given as follows:

f_K (L, R) = (L XOR f(R, key), R)

Where L, R be the left 4-bits and right 4-bits of the input, XOR is the exclusive-OR operation and key is a sub -key.

Computation of f(R, key) is done as follows.

i. Apply expansion/permutation
E/P= [4 1 2 3 2 3 4 1] to input 4-bits.

ii. Add the 8-bit key (XOR).

iii. Pass the left 4-bits through S-Box S0 and the right 4-bits through S-Box S1.

iv. Apply permutation P4 = [2 4 3 1].

The two S-boxes are defined as follows:

S0

$$\begin{pmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{pmatrix}$$

S1

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{pmatrix}$$

The S-boxes operate as follows: The first and fourth input bits are treated as 2-bit numbers that specify a

row of the S-box and the second and third input bits specify a column of the S box. The entry in that row and column in base 2 is the 2-bit output.

3. The Switch Function (SW):

Since the function f_K allows only the leftmost 4-bits of the input, the switch function (SW) interchanges the left and right 4-bits so that the second instance of f_K operates on different 4-bits. In this second instance, the E/P, S0, S1 and P4 functions are the same as above but the key input is K2.

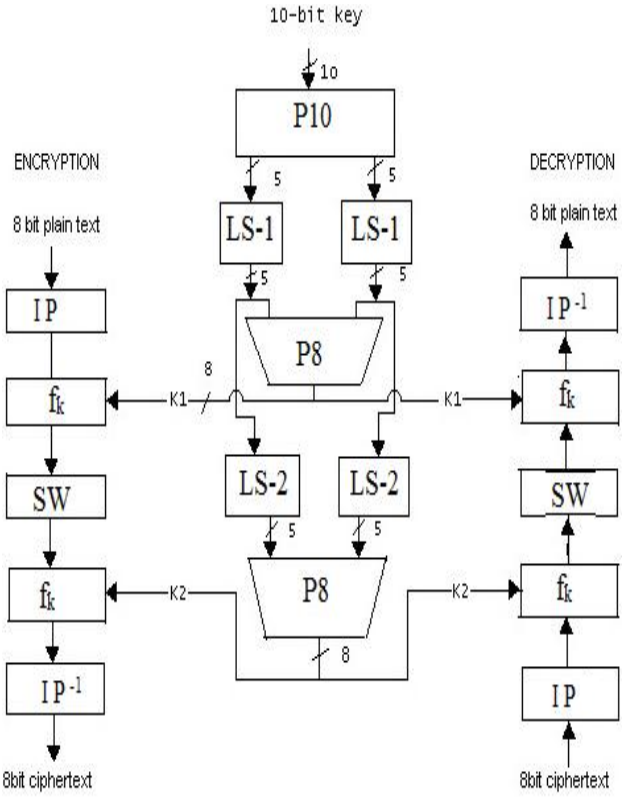

Fig 1: S-DES Encryption Key Generation and Decryption Algorithm

2.1.3 Decryption Algorithm

The decryption is the reverse process of encryption. It takes 8 bit ciphertext, 10 bit key and generates 8 bit plaintext. The sequence of blocks used in decryption is shown in Figure 1. The functions of these blocks are same as that used in encryption side. The order of the key used at the decryption side is reversed i.e., Key K2 is used first then key K1 is used. The computational time required for decryption is same as that of encryption. The more details about S-DES encryption, key generation and decryption algorithm can be found detail in [2, 3].

2.2 Basics of Computational Intelligence

This section briefly explains GA, PSO and GSO.

2.2.1 Genetic Algorithm

GA provides robust searches in complex spaces. These algorithms are computationally simple. A simple genetic algorithm that yields good results in many practical problems is composed of three operators: Selection (Reproduction), Crossover and Mutation [17-20]. Selection strategies determine which chromosome will take part in the evolution process. The different Selection strategies are Population Decimation, Proportionate selection and Tournament Selection [20].

After the selection, the next operation is mating scheme. Selection Strategies are involved with selecting which individuals will take part in the evolution process, the mating scheme will select which two parent chromosomes will mate with one another. The mating scheme that exists includes Best-Mates Worst, Adjacent fitness Pairing and Emperor Selective Mating. In Best-Mates worst mating, as the name indicates the chromosome with the highest fitness mates with the chromosome with the lowest fitness which is the preferred mating technique.

The next operation is Crossover, which selects genes from parent chromosome and creates a new offspring. This operator randomly selects some crossover point and everything before this point copy from a first parent and everything after this point copy from the second parent. The two newly generated chromosomes may be better than their parent chromosome and the evolution process may continue. Crossover is continued by Mutation. This operator randomly changes one or more bits in the Chromosome. The purpose of this operator is to prevent the population to escape from minimum value. The mutation is carried out according to the mutation probability Pmutation. Mutation rate must be low. A best mutation rate is 0.015(i.e., 1.5%).

2.2.2 Particle Swarm Optimization

Swarm Intelligence is an innovative paradigm used for solving the complicated problems. PSO is a population based optimization tool which could be implemented and applied to solve various optimization problems [4].

The Canonical PSO model consists of a swarm of particles, which are initialized with a population of random candidate solutions [4]. They move iteratively through the d-dimension problem space to search the new solutions, where the cost C_k , can be calculated as the certain quality measure. Each

particle has a position represented by a position-vector $\mathbf{x_i}$ (i is the index of the particle), and a velocity represented by a velocity-vector $\mathbf{v_i}$. Each particle remembers its own best position so far in the vector $\mathbf{x_i}$ # (pbest), and its j-th dimensional value is $\mathbf{x_{ij}}$#. The best position-vector among the swarm so far is then stored in a vector x*(gbest), and its j-th dimensional value is x_j*. During the iteration time t, previous velocity ($v_{ij}(t)$) is updated to the new velocity (v_{ij} j(t+1)), determined by Eq.(1). The new position is then determined by the sum of the previous position and the new velocity, as given by Eq(2).

$$\mathbf{v_{ij}(t+1)} = w.\mathbf{v_{ij}(t)} + c_1r_1(\mathbf{x_{ij}}^{\#}\mathbf{(t)} - \mathbf{x_{ij}(t)}) + c_2r_2(x_j^*(t) - \mathbf{x_{ij}(t)}). \quad \mathbf{(1)}$$

$$\mathbf{x_{ij}(t+1)} = \mathbf{x_{ij}(t)} + \mathbf{v_{ij}(t+1)}. \quad \mathbf{(2)}$$

Where 'w' is called as the inertia factor, r1 and r2 are the random numbers, which are used to maintain the diversity of the population and are uniformly distributed in the interval [0,1] for the j-th dimension of the i-th particle. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as co-efficient of the social component. From Eq. (1), a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm.

2.2.2.1 Binary PSO

The canonical PSO is basically developed for continuous optimization problems. In our case, the cryptanalysis problem deals with binary information. Hence the canonical PSO cannot be applied directly. In our problem, the variable x_{ij} represents the key in binary form, hence x_{ij} (t) should take 0 or 1, but from the equation (1), we can see that the results of v_{ij} (t+1) may not be an integral, and from equation (2) we can see x_{ij} (t+1) may take numerical other than 0,1 after iteration. The equations have to be adjusted in such a way that the velocity and position are to be in binary form. In the binary PSO, we can define a particle's position and velocity in terms of changes of probabilities that will be in one state or the other [15, 16]. At each time step, each particle updates its velocity and moves to a new position according to the Eq.(3) and (4):

$$\mathbf{v_{ij}(t+1)} = w.\mathbf{v_{ij}(t)} + c_1r_1(\mathbf{x_{ij}}^{\#}\mathbf{(t)} - \mathbf{x_{ij}(t)}) + c_2r_2(x_j^*(t) - \mathbf{x_{ij}(t)}) \quad \mathbf{(3)}$$

$$\mathbf{x_i(t+1)} = \mathbf{1} \quad \mathbf{if} \quad \rho \le \mathbf{s(v_i(t))},$$

$$\mathbf{0} \quad \mathbf{otherwise.} \quad \mathbf{(4)}$$

Where 'ρ' is a random function in the closed interval [0, 1]. The Velocity update equation is similar to that of canonical PSO. The only difference is in the position update as given in the equation (4).

2.2.3 Genetic Swarm Optimization

Genetic Swarm Optimization is a hybrid evolutionary technique that combines the effectiveness of GA and PSO and overcome the problem of premature convergence.

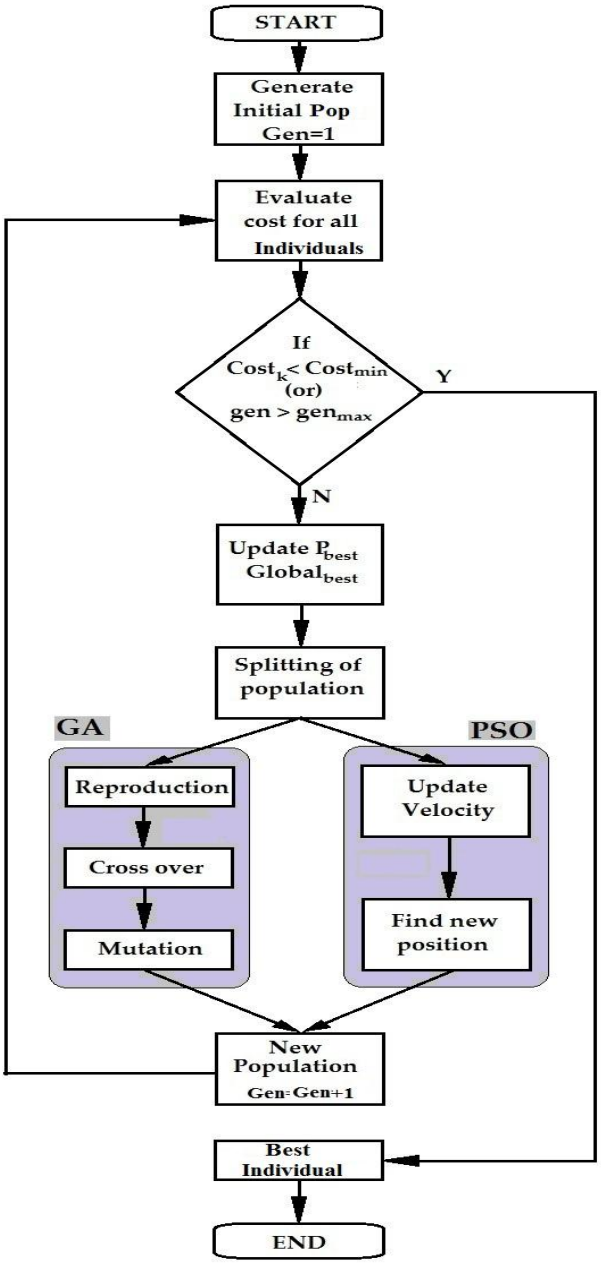

Fig 2: A GSO Cycle for cryptanalysis

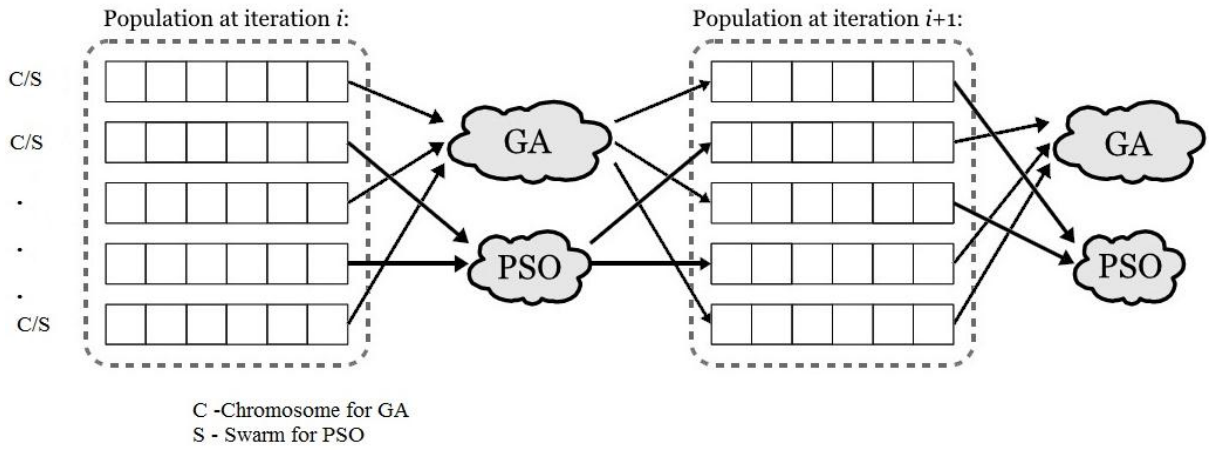

Fig 3: Evolutionary process in GSO

GSO have strong co-operation of GA and PSO, since it maintains the integration of these two techniques for the entire run. In fact, this kind of updating technique yields a particular evolutionary process where individuals not only improve their score for natural selection of the Cost or for good-knowledge sharing, but for both of them at the same time. The basics of GSO are discussed in [13]. The working principle of GSO is given below:

In each iteration, the population is randomly divided into two parts. Each part is taken as the population for GA and PSO respectively and cost is computed for the new set of population. The populations were recombined in the updated population which is again divided into two parts in the next iteration for the next run of GA or PSO. This process continues until the Cost value is converged. These processes were combined to form GSO cycle and shown in Figure 2.

An important parameter called hybridization coefficient h_{coeff} drives the GSO algorithm where h_{coeff} is expressed in terms of percentage of the population. In each iteration h_{coeff} percentage of the total population is processed by GA and the remaining (1- h_{coeff}) percentage of the total population is processed by PSO. For example, if h_{coeff} = 0, then the whole population is processed by PSO operators, i.e., the algorithm becomes purely PSO and if h_{coeff} =1, then the whole population is processed by GA operators, i.e., the algorithm becomes purely GA. While 0< h_{coeff} <1 means the corresponding percentage of h_{coeff} population is processed by GA and the remaining population is processed by PSO. The population evolution using GSO algorithm is shown in Figure 3.

The effectiveness of GSO depends upon the parameters selected in GA and PSO and also the hybridization coefficient, which can be static or dynamic. In case of static hybridization, h_{coeff} is fixed (say 0.2) , whereas for dynamic hybridization, h_{coeff} is taken randomly for each iteration as shown in Table 1.

Table 1 : Hybridization Coefficient during iterations

Static Hybridization $h_{coeff}(k)$	0.2 for all k
Dynamic Hybridization $h_{coeff}(k)$	Rand( ) k=1 Rand( ) k=2 …. …. …. ….. Rand( ) k=iteration Number

3. Problem Formulation

In this section, we describe the Cost function used and our proposed approach to illustrate the effectiveness of our algorithm. The goal of the problem is to minimize the cost function.

3.1 Cost Function

The main task in formulating the cryptanalysis problem using CI is to find out the effective cost function. Finding the appropriate cost function is a difficult task. Once the effective cost function is defined, then the problem becomes facile. The Cost function used for the considered problem is given by the equation (5), which represents the ngram[1] statistics of the decrypted message and the language is assumed to be known.

$$\textbf{Cost } C_k = \alpha \Sigma(i \varepsilon \tilde{A}) |K(i)u - D(i)u| + \beta \Sigma(i, j \varepsilon \tilde{A})| K(i, j)b - D(i, j)b| + \gamma \Sigma(i, j, k \varepsilon \tilde{A})|K(i, j, k)t - D(i, j, k)t| \quad (5)$$

where Ã denotes the language alphabet {a, b…z, _} for English (where _ represents the space symbol); K and D are the known language statistics and decrypted message statistics respectively; u, b, and t denote the unigram, digram and trigram statistics, respectively. For example the known statistics for the frequency of occurrence of the letter 'e' in an English text is 12.7% while for 't' it is 9.1%, the digram frequency for 'th' is 3.21% while 'he' it is 3.05%. The frequency statistics for other digrams and trigrams are found in [14]. The statistics 'u' can be computed by counting the number of occurrence of each character and divide it by the total number of characters. In the same way, the two letter combination and three letter combination were counted and divided by total number of characters that gives 'b' and 't' respectively. In view of the computational complexity of n-gram statistics, all unigrams were useful and very few digrams and trigrams were useful. Among the possible 27^1 unigrams, 27^2 digrams and 27^3trigrams, the considered unigrams, digrams and trigrams are reported in table 2.

Finally α, β and γ are the weights used for assigning different priorities to each of the three statistics and $\alpha+ \beta + \gamma = 1$. The values of α, β and γ are 0.2, 0.4 and 0.4. Since very few digrams and trigrams are considered, more weight is assigned to β and γ.

Table 2: Useful Unigrams, digrams and trigrams

Unigrams	a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w, x,y,z, _(space).
Digrams	th, he, in, er, an, re, ed, on, es, at, to, nt, nd, ha, ea, ou, is, it, ti, et, ar, te, se, hi, of, as, or
Trigrams	the, ing, her, ere, and, tha, was, for, ion, has, men, nce.

The objective of the problem is to minimize the cost function. To set the minimum cost value ($cost_{min}$), a standard English text file from novels and cryptography book were taken with different sizes. The cost value is computed for those standard files and the average of the cost value is taken which gives $cost_{min}$.

3.2 Attacking the key using GA

In this sub section, we describe how GA can be used to break the key used in SDES. Before that we relate some important terms used in GA that makes some sense in cryptanalysis is shown in table 3.

Table 3:Parameters that relate GA and Cryptanalysis

Parameter	**GA**	**GA in Cryptanalysis**
Gene	A single bit in chromosome	A single bit in key
Chromosome	Any Possible Solution	Any Possible key
Population	Group of Chromosomes	Group of keys
Cost Value	A function to evaluate the performance	Letter Frequency Analysis
Generations	Number of generation	Number of Iterations

After relating the parameters in GA and cryptanalysis, the following operations were performed to carry out the cryptanalysis of S-DES using GA in order to break the key.

1. Initial keys were generated randomly. The number of keys considered initially represents the population size. The results show that it is better to consider low population size and increase the number of generations. So that the crossover rate will be high.
2. Using the randomly generated keys, decrypt the known ciphertext to generate the plaintext, and compute the cost C_k using the equation (5).
3. The computed cost value is compared with the predefined minimum cost $cost_{min}$. If the computed cost is less than or equal to the $cost_{min}$, we can conclude that the corresponding key with the minimum cost is the optimum key and go to step 11.
4. If the condition in step 3 is not met, then apply GA parameters.
5. Select the parent keys to generate a new set of children keys using the selection strategies.
6. Do the mating among parent keys.
7. Do the crossover. Random point crossover is preferred.
8. Perform the mutation operation to the current population and generate a new set of keys.
9. For the newly generated keys, compute the Cost function and go to step 3.

10. Repeat the step 2 to 9 until the Cost is minimized or the maximum number of generation is reached. If the maximum number of generation is reached then the key with the maximum Cost in the final generation corresponds to the optimum key.
11. Display the Optimum Key and terminate the process.

These processes were combined to form a GA cycle and shown in figure 4.

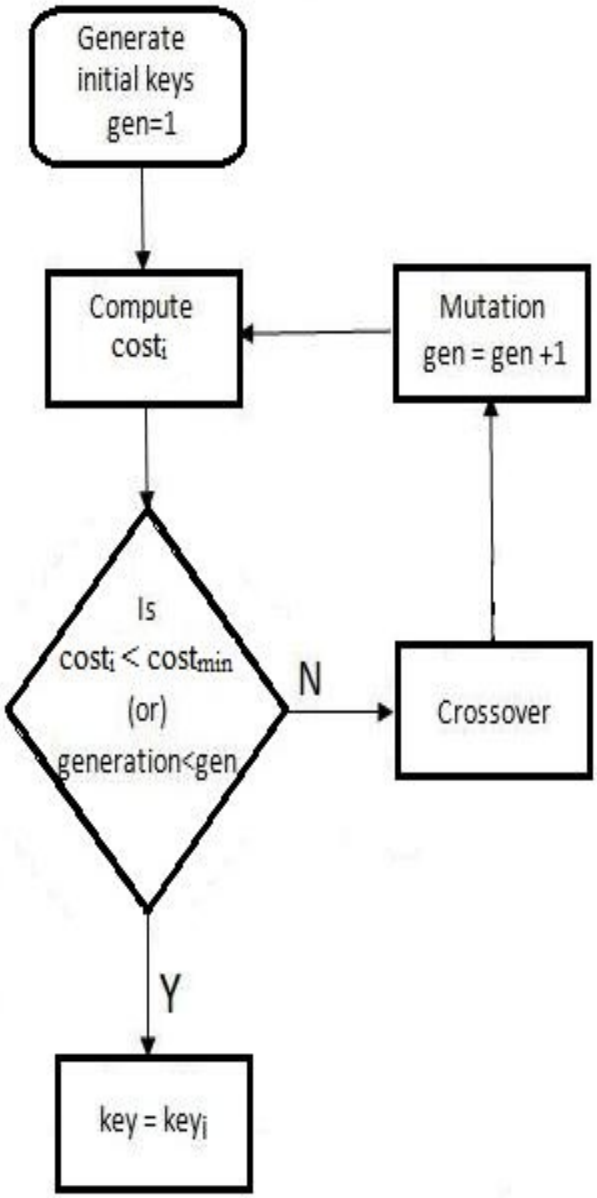

Fig 4: A genetic Algorithm Cycle for Cryptanalysis.

3.3 Attacking the key using PSO

In this section, we describe our proposed approach and illustrate how PSO can be applied to break the key in the field of cryptanalysis. In a swarm of particles, each particle represents a key, which is a 10 bit binary key. Initialize the swarm Particles X_i. Using the generated particles, decrypt the known cipher text to obtain the plaintext and evaluate the cost function from the obtained plain text by computing the letter frequency analysis i.e., by equation (5). The best position is associated with the minimum cost value i.e., Cost (Pibest) of the particle Pibest and Global best (Pgbest) is the best position among all particles in the swarm which is achieved so far. The global position is associated with the global cost value, Cost (Pgbest) of the particle Pgbest

Velocity and Particle's position are updated according to the equation (3) and (4).The cost is computed from the updated particle's position in order to update the position of Pibest and Pgbest. The process is continued until the cost function is minimized or maximum number of iteration is reached. If there is no improvement in the cost for some iteration continuously then the algorithm is stopped. Algorithm for finding the key using PSO is shown in Table 4.

Table 4: Algorithm for Cryptanalysis of S-DES using PSO

1. Set
   i)Number of iterations
   (ii) Minimum cost -$Cost_{min}$.
2. Initialize the swarm Particles
3. Decrypt known Cipher text using generated particles.
4. Compute Cost Value C_k.
5. Update the Velocity and Particle's position according to the equations (3) and (4)
6. Update the position of gbests and pbests.
   If Cost ($X_i(t)$) < Cost(P_{ibest}) then P_{ibest}= $X_i(t)$
   if Cost ($X_i(t)$) < Cost(P_{gbest}) then P_{gbest}= $X_i(t)$
7. Check for Stopping Criteria. Repeat steps 3-6 until the stopping criteria are satisfied.
8. Display Key found: Key = $P_{gbest.}$

3.4 Genetic Swarm Optimization

In this section, the proposed novel approach GSO to attack DES is explained. Initialize the population randomly and select the hybridization coefficient. Take the known ciphertext file and decrypt it using initial population (keys) and compute the cost value using the equation (5) for the entire population and check for the minimum cost value. If the computed cost is less than $cost_{min}$ then the key with the corresponding cost is the actual key. Continue the evolution process using GSO algorithm (as explained in section 2.2.3) until the stopping criteria are met. The algorithm for GSO for cryptanalysis is shown in table 5. For GSO, we take the hybridization coefficient as static and dynamic, as already described in table 1. For static case, two different values 0.2 and 0.3 were taken and processed. For example h_{coeff} (k) = 0.3; i.e., 30% of the population were processed by GA and the remaining 70% of the population were processed by PSO to generate new population.

Table 5: Algorithm for Cryptanalysis of S-DES using GSO

1. Initialize POP_{GSO} Randomly
2. Select hcoeff : i.e., Static or Dynamic
   If Static: Case i) $h_{coeff}(k)= 0.2$ for all k
   Case ii) $h_{coeff}(k) =0.3$. for all k
   If dynamic : $h_{coeff}(k) = rand( )$
   $POP_{GA} =h_{coeff}(k) * P_{GSO}$ individuals for GA
   $POP_{PSO} =(1-h_{coeff}(k))* POP_{GSO}$ individuals for PSO
   Where 'k' is the generation number
3. For POP_{GA} : Generate new population by Applying GA
   For POP_{PSO} : Generate new population by applying PSO
4. Update New population
   $POP_{GSO}= POP_{GA +} POP_{PSO}$
5. Compute the cost for POP_{GSO}
6. Check for stopping criteria. Repeat steps 2-5 until stopping criteria is satisfied.
7. If stopping criteria are met then Key =$Key_{optimum.}$

4. Experimental Set up and Results:

The proposed algorithm was implemented using Matlab on an Intel Core^{i3} processor. Our objective is to analyse the performances of GA, PSO and GSO in attacking the ciphertext. Different ciphertext files of various sizes were considered. The ciphertexts were constructed from Standard English novels and Cryptography book for technical text using SDES encryption.

The total number of generation depends upon the initial population size. The initial population and generation is taken in such a way that the total key search space is set to 300 , in order to keep the search space less compared to Brute-force search space atleast by a factor of 3. For instance, if the initial population size is taken as 10 then the number of generations is 30.

The parameters used for the GA based cryptanalysis and PSO based cryptanalysis is shown in table 6 and table 7 respectively. For GSO, the GA parameters and PSO parameters were taken from the table 6 and 7 respectively. In table 8, the hybridization coefficients used for GSO based cryptanalysis is shown. Instead of analysing GA and PSO separately, we consider both of them as a special case of GSO with appropriate hybridization coefficient as already explained in section 2.2.3. That is, if h_{coeff} (k) = 1 then GSO is equivalent to GA and if $h_{coeff}(k)= 0$ then GSO is equivalent to PSO. Table 8 shows the different values for $h_{coeff}(k)$ and the results show that GSO performs better when compared to GA and PSO.

Table 6: GA Parameters

Population Size - 10
Number of generations - 30
Mating Scheme – Best Mate Worst
Crossover Type - Random
Mutation Rate - 0.015

Table 7: Parameters for PSO

Self-Recognition Parameter c1	1
Social Parameter c2	4-c1
Constriction parameter C	1
Inertia weight (w)	0.99 < w < 0
Initial Population	10
Number of Iterations	30

Table 8: Comparison results for GA, PSO and GSO

Key Used (10 bits)	Key Found	Key Search Space (Number of Ciphertext =1000 characters)				
		GSO $h_{coeff}(k)=1$ (GA)	GSO $h_{coeff}(k)=0$ (PSO)	GSO $h_{coeff}(k)=0.2$	GSO $h_{coeff}(k)= 0.3$	GSO $h_{coeff}(k)=rand( )$
02DE	02DE	247	202	212	198	189
01BF	01BF	237	209	196	184	177
0037	0037	242	217	204	196	183
00AF	00AF	234	212	192	186	179
Average Key Search		240	210	201	191	182

Initially the random keys are generated and the known ciphertext are decrypted using these random keys. The cost value is computed for all the keys using equation 5. In case of GA i.e., GSO with $h_{coeff}(k)=1$, the key is recovered in 24 generations. The average number of keys searched is 240 and the search space is reduced by the factor of 4.3 when compared to Brute-force attack (where the average number of keys searched is 1024). This can be observed in Table 8.

When $h_{coeff}(k)= 0$ i.e., in case of PSO , the average key search is 210 and the search space is reduced by a factor of 4.8 when compared to Brute-force attack and when compared to GA, the average key search is reduced by the factor of 1.1.

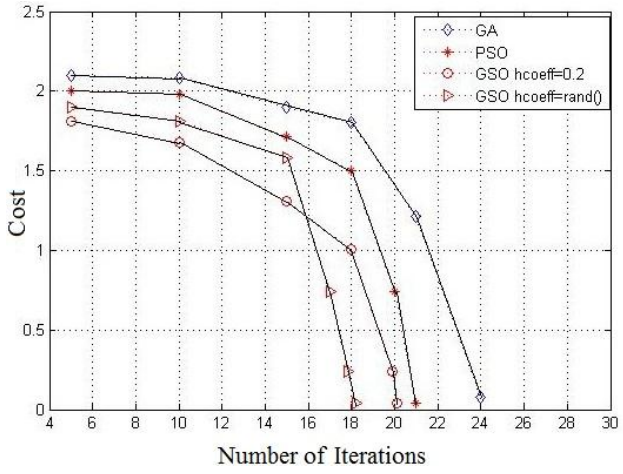

Figure 5: Cost Vs Number of Iterations

If $h_{coeff}(k)$ is set to 0.2 then the average number of keys searched is 201 and $h_{coeff}(k)$ is set to 0.3 then the average number of keys searched is 191 The performance of GSO is improved if $h_{coeff}(k)$ is dynamic In this case the average key search is 182 where the key search space is reduced by a factor of 5.63 which is estimable reduction factor in cryptanalysis.

If S-DES decryption is done in 1 second, the average time required to attack the ciphers using GSO is nearly 180 seconds (3minutes), whereas in case of PSO and GA the time required for convergence of the algorithm is 210 seconds and 240 seconds respectively.

Figure 5 illustrates for varying hybridization coefficient, how the cost value converges when the input file size is 1000 ciphertext characters. If the hybridization coefficient is dynamic then the convergence rate is very high thereby reducing the key search space.

The experimental results show that GSO algorithm is independent of the initial keys considered i.e., initial seed. Additionally, the convergence of the algorithm depends on the size of the considered cipher text. If the size of the ciphertext is small, the decrypted plain text contains little information about the letter frequency and when the size of the ciphertext is large, the algorithm converges fastly since more letter frequency information is available and the key is recovered fastly with less number of generations. This can be observed in table9.

Table 9: Effect of Key search space with size of Ciphertexts

Number of Ciphertexts	Average Key Search Space				
	GSO $h_{coeff}(k)=1$ (GA)	GSO $h_{coeff}(k)=0$ (PSO)	GSO $h_{coeff}(k)=0.2$	GSO $h_{coeff}(k)= 0.3$	GSO $h_{coeff}(k)=rand(k)$
100	272	258	242	235	221
500	265	232	228	205	197
1000	240	210	201	191	182

5. Conclusion

In this paper, a novel approach GSO by combining the effectiveness of GA and PSO is proposed to attack Simplified-DES. From the results and analysis, it is observed that GSO reduces the key search space by the factor of 5.6 and runs through less time. Implementing our approach in high speed computers further reduces the time consumption. This shows that the GSO can be effectively used in the field of cryptanalysis and this approach has been reported for the first time to attack the ciphers. Though SDES is simpler than DES, this gives a better idea to attack DES and other complex block ciphers like AES .

References:

[1] Neal Koblitz, A course in Number Theory and Cryptography, Springer International Edition, 2008.

[2] William Stallings, Cryptography and Network Security Principles and Practices, Pearson Education, 2004.

[3] Behrouz A. Forouzan, Cryptography and Network Security, Tata McGraw hill Education, 2nd edition 2008.

[4] Nadia Nedjah, Ajith Abraham, Luzia de Macedo Mourelle, Swarm Intelligent systems, Studies in Computational Intelligence, Vol.26,2006.

[5] Nadia Nedjah, Ajith Abraham, Luzia de Macedo Mourelle, Computational Intelligence in Information Assurance and Security, Studies in Computational Intellige nce, Vol. 57,2007.

[6] Spillman R,Janssen M, Nelson B and Kepner N, "Use of Genetic Algorithm in Cryptanalysis of Simple Substitution Cipher" Cryptologia, Vol.17, No.4, pp. 367-377, 1993.

[7] Garg Poonam, A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm, International Journal of Network Security & Its Applications (IJNSA), Vol.1, No 1, April 2009 pp-34-42.

[8] Garg Poonam , Cryptanalysis of SDES via Evolutionary Computation Techniques, International Journal of Computer Science and Information Security, Vol. 1, No. 1, May 2009. Pp 117-123.

[9] Nalini, Attacks of simple block ciphers via efficient heuristics, Information Sciences, pp 2553-2569.

[10] Nalini, Cryptanalysis of Simplified data encryption standard via Optimization heuristics, International Journal of Computer Sciences and network security, vol 6, No 1B, Jan 2006.

[11] Vimalathithan.R, M.L.Valarmathi, "Cryptanalysis of S-DES Using Genetic Algorithm", International Journal of Recent Trends in Engineering, Vol2, No.4, November 2009, pp.76-79.

[12] Vimalathithan.R, M.L.Valarmathi, "Cryptanalysis of S-DES Using Particle Swarm Optimization", 10th National Workshop on Cryptology, Coimbatore, India, Sep 2010.

[13] A. Gandelli, F. Grimaccia, M. Mussetta, P. Pirinoli, R.E. Zich, "Development and Validation of Different Hybridization Strategies between GA and PSO", Proc. of the 2007 IEEE Congress on Evolutionary Computation, Sept. 2007, Singapore, pp. 2782–2787.

[14] A Menezes,P.Vanoorschoot,S.Vanstone Handbook of Applied Cryptography,CRC Press,1996.

[15] J,kennedy, R.Eberhart, "A discrete Binary version of the Particle Swarm Algorithm," International Conference on Neural network, Vol. IV,pp:4104-4108, Australia,1997.

[16] J,kennedy, R.Eberhart, "Particle Swarm Optimization," IEEE international Conference on Neural Networks,pp:1942-1948,Australia,1995

[17] Collin R.Reeves, J,E Rowe, Genetic Algorithms-Principles and Perspectives, A guide to GA theory, Kluwer Academic Publishers.

[18] Haupt R.L, Haupt S.E. , Practical Genetic Algorithms, 2nd ed., Wiley, 2004.

[19] M.Mitchell, An Introduction to Genetic Algorithms, First MIT press paperback edition, 1998.

[20] Goldberg D.E., "Genetic Algorithm in Search, Optimization and Machine Learning", Boston, Addison-Wesley, 1999.