# Attack on the Polyalphabetic Substitution Cipher Using a Parallel Genetic Algorithm

A. Dimovski[1], D. Gligoroski[2]

*Abstract* – **In this paper is presented an automated attack on the polyalphabetic substitution cipher. The property which make this cipher vulnerable, is that it is not sophisticated enough to hide the inherent properties or statistics of the language of the plaintext. The attack described here effectively reduces the complexity of a polyalphabetic substitution cipher attack to that of a monoalphabetic one, if there is a computer with *B* processing nodes, where *B* is the period of the polyalphabetic substitution cipher.**

*Keywords* – **Polyalphabetic substitution cipher, Cryptanalysis, Parallel genetic algorithm**

## I. Polyalphabetic Substitution Ciphers

The polyalphabetic substitution cipher is a simple extension of the monoalphabetic one. The difference is that the message is broken into blocks of equal length, say $B$, and then each position in the block $(1, \ldots, B)$ is encrypted (or decrypted) using a different simple substitution cipher key. The block size $B$ is often referred to as the period of the cipher.

An example of a polyalphabetic substitution cipher is shown on Table 1. The block size (i.e., $B$) is chosen to be three, and Table 1 gives an example key and shows the corresponding encryption, it is clear that the decryption process is reversal of the encryption.

Table 1. Example of the polyalphabetic substitution cipher key and encryption process

| KEY: | |
|---|---|
| Plaintext: | |
| ABCDEFGHIJKLMNOPQRSTUVWXYZ_ | |
| Ciphertext: | |
| PQOWIEURYTLAKSJDHFGMZNX_BCV | (Position 1) |
| LP_MKONJIBHUVGYCFTXDRZSEAWQ | (Position 2) |
| GFTYHBVCDRUJNXSEIKM_ZAWOLQP | (Position 3) |
| ENCRYPTION: | |
| Position: | 12312312312 |
| Plaintext: | HOW_ARE_YOU |
| Ciphertext | RYWVLKIQLJR |

The number of possible keys for a polyalphabetic substitution cipher using an alphabet size of 27 and a block size of $B$ is $27!^{B}$. This is significantly greater than the simple substitution cipher with 27! possible keys, especially when the period $B$ is large. The polyalphabetic substitution cipher is somewhat more difficult to cryptanalyse than the simple substitution cipher because of the independent keys used to encrypt successive characters in the plaintext, but it is still relatively simple to cryptanalyse the polyalphabetic substitution cipher based on the $n$-gram statistics of the plaintext language.

So, despite the monoalphabetic substitution cipher where every bigram (for example _A) is mapped to the same encrypted bigram each time, this is not the case for the polyalphabetic substitution cipher, where the encrypted value of a bigram is dependent upon two factors: the individual key values and the position of the characters within the block.

The polyalphabetic substitution cipher is simply a number of simple substitution ciphers operating on the different positions within each block. One possible attack strategy, then, is to solve each of the simple substitution ciphers in parallel. Here we will use a parallel genetic algorithm to attack the polyalphabetic substitution cipher.

## II. Genetic Algorithms

The genetic algorithm is based upon Darwinian evolution theory. The genetic algorithm is modelled on a relatively simple interpretation of the evolutionary process, however, it has proven to be a reliable and powerful optimisation technique in a wide variety of applications.

Holland [1] in 1975, was first who proposed the use of genetic algorithms for problem solving. Goldberg [2] and DeJong [3] were also pioneers in the area of applying genetic processes to optimisation. Over the past twenty years numerous applications and adaptations of genetic algorithms have appeared in the literature.

Consider a pool of genes that have the ability to reproduce, are able to adapt to environmental changes and, depending on their individual strengths, have varying lifespans. In such an environment only the fittest will survive and reproduce giving, over time, genes that are stronger and more resilient to conditional changes. After a certain amount of time the surviving genes could be considered "optimal" in some sense. This is the model used by the genetic algorithm, where the gene is the representation of a solution to the problem being optimised.

As with any optimisation technique there must be a method of assessing each solution. The assessment technique used by a genetic algorithm is usually referred to as the "fitness function". The aim is always to maximise the fitness of

[1]Faculty of Natural Sciences and Mathematics, Ss. Cyril and Methodius University Arhimedova b.b., PO Box 162, 1000 Skopje, Macedonia adimovski@ii.edu.mk

[2]Faculty of Natural Sciences and Mathematics, Ss. Cyril and Methodius University Arhimedova b.b., PO Box 162, 1000 Skopje, Macedonia gligoroski@yahoo.com

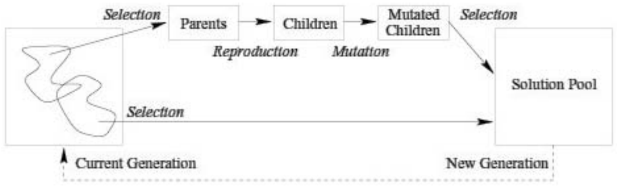

Fig. 1. The Evolutionary Process.

the solutions in the solution pool.

Fig. 1 gives an indication of the evolutionary processes used by the genetic algorithm. During each iteration of the algorithm the processes of selection, reproduction and mutation each take place in order to produce the next generation of solutions. The actual method used to perform each of these operations is very much dependent upon the problem being solved and the representation of the solution.

An algorithmic representation of the genetic algorithm is given in Fig. 2. This description is independent of any solution representation, fitness function, selection scheme, reproduction scheme and mutation scheme. Each of these will be described in detail where the genetic algorithm has been applied.

```
1. Initialize algorithm variables: G the maximum number of generations to consider, M the solution pool size and any other problem dependent variables.

2. Generate an initial solution pool containing M candidate solutions. This initial pool can be generated randomly or by using a simple known heuristic for generating solutions to the problem at hand. This solution pool is now referred to as the *current* solution pool.

3. For G iterations, using the current pool:

    a) Select a breeding pool from the current solution pool and make pairings of parents.
    b) For each parental pairing, generate a pair of children using a suitable mating function.
    c) Apply a mutation operation to each of the newly created children.
    d) Evaluate the fitness function for each of the children.
    e) Based on the fitness of each of the children and the fitness of each of the solutions in the current pool, decide which solutions will be placed in the new solution pool. Copy the chosen solutions into the new solution pool.
    f) Replace the current solution pool with the new one. So, the new solution pool becomes the current one.

4. Choose the fittest solution of the final generation as the best solution.
```

Fig. 2. The Genetic Algorithm

## III. A Parallel Genetic Algorithm Attack

We will attack the polyalphabetic substitution cipher using a number of genetic algorithms running in parallel, each solving a different part of the problem. Fig. 3 is a pictorial representation of this strategy with $M$GA's running in parallel and communicating every $k$iterations.

Let's consider a polyalphabetic substitution cipher consisting of $M$ monoalphabetic or simple substitution ciphers. There will then be $M$ genetic algorithms (call them GA1, GA2, . . . , GA$M$) solving each of the $M$simple substitution ciphers. GA$j$ $(1 < j < B)$, which is attempting to find the key to the cipher of position $j$, in determining the cost of each of the solutions in its pool, GA$j$ uses the current best key from each of its neighbours to find the bigram and trigram statistics.

Before implementing the parallel attack a number of design problems have to be solved.

### A. *Suitability Assessment*

The technique used to compare candidate keys is to compare $n$-gram statistics of the decrypted message with those of the language (which are assumed known). Equation 1 is a general formula used to determine the suitability of a proposed key $(k)$. Here, $A$ denotes the language alphabet (i.e., for English, [A, . . . , Z, _], where _ represents the space symbol), $K$ and $D$ denote known language statistics and decrypted message statistics, respectively, and the indices $u$, $b$ and $t$ denote the unigram, bigram and trigram statistics, respectively. The values of $\alpha$, $\beta$ and $\gamma$ allow assigning of different weights to each of the three $n$-gram types.

$$C_k = \alpha \cdot \sum_{i \in A} \left| K^u_{(i)} - D^u_{(i)} \right| + \beta \cdot \sum_{i,j \in A} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| + \gamma \cdot \sum_{i,j,k \in A} \left| K^t_{(i,j,k)} - D^t_{(i,j,k)} \right| \quad (1)$$

Spillman [4], in his attack on the simple substitution cipher use Eq. (2). This equation is based on unigram and bigram statistics.

$$C_k \approx \alpha \cdot \sum_{i \in A} \left| K^u_{(i)} - D^u_{(i)} \right| + \beta \cdot \sum_{i,j \in A} \left| K^b_{(i,j)} - D^b_{(i,j)} \right| \quad (2)$$

The only difference between these assessment functions is the inclusion of different statistics. In general, the larger the $n$-grams, the more accurate the assessment is likely to be. It is usually an expensive task to calculate the trigram statistics - this is, perhaps, why they are omitted in Eq. (2). The complexity of determining the fitness is $O(N^3)$ (where $N$ is the alphabet size) when trigram statistics are being determined, compared with $O(N^2)$ when bigrams are the largest statistics being used.

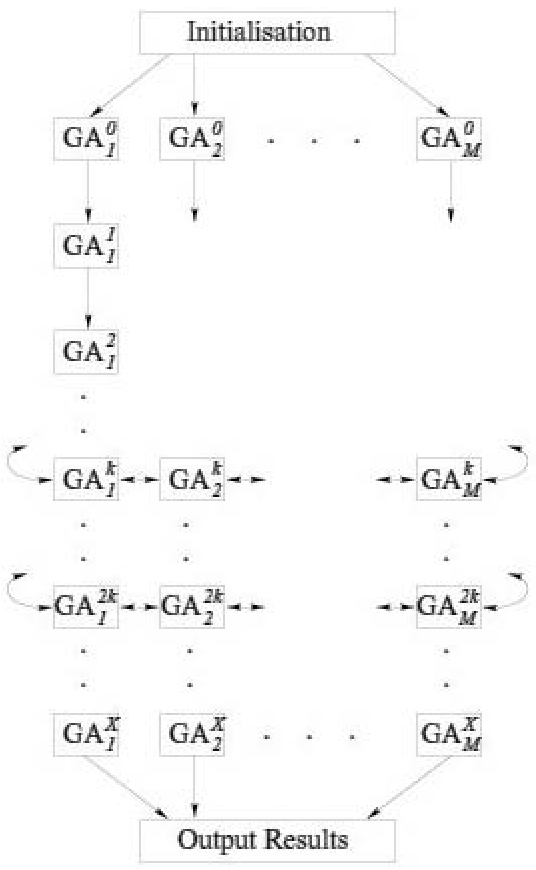

Fig. 3. A parallel genetic algorithm

In the case of the polyalphabetic substitution cipher, without knowledge of the keys for the two adjacent block positions it is impossible to determine bigram or trigram statistics. To overcome this problem the following strategy is used:

1. Initially only unigram statistics are used in determining the cost of the solutions in any pool.

2. Every $k$ iterations of each GA, the most fit solution in the current pool is sent to each of the neighbouring GA's. Each GA has knowledge of the entire ciphertext message so it is able to determine a fitness based on unigram, bigram and trigram statistics using ciphertext characters in its position, the position to the left and the position to the right.

## B. *The Reproduction Process*

The mating function utilised here is similar to the one proposed by Spillman [4], who use a special ordering of the key. The characters in the key string are ordered such that the most frequent character in the ciphertext is mapped to the first element of the key (upon decryption), the second most frequent character in the ciphertext is mapped to the second element of the key, and so on. The reason for this ordering will become apparent upon inspection of the mating function. For example, the key FGHIJKLMNOPQRSTUV_WXYZAB indicates that the most frequent character in the ciphertext represents a plaintext F; the second most frequent character in the ciphertext represents a plaintext G, etc.

Given two parents constructed in the manner just described, the first element of the first child is chosen to be the one of the first two elements in each of the parents, which is most frequent in the known language statistics. This process continues in a left to right direction along each of the parents to create the first child only. If, at any stage, a selection is made which already appears in the child being constructed, the second choice is used. If both of the characters in the parents for a given key position already appear in the child then a character is chosen at random from the set of characters that do not already appear in the newly constructed child.

The second child is formed in a similar manner, except that the direction of creation is from right to left and, in this case, the least frequent of the two parent elements is chosen.

```
Parent 1
        FGHIJKLMNOPQRSTUV_WXYZABCDE
Parent 2
        CDEFGHIJKLMNOPQRSTUVWXY_ZAB

Child 1
        CDEIGHLMNOPQRSTUV_WXYZABFJK
Child 2
        CIEFGHLJKLMNOPQRSTUVWXY_ZDB
```

Fig. 4. The mating process

## C. *Description of the Algorithm*

The implementation of each genetic algorithm proceeds as follows:

1. Each GA is given language statistics for unigrams, bigrams and trigrams, the ciphertext, the block size ($B$) and this GA's position within the block, $j$ ($1 \leq j \leq B$), the frequency of inter-GA communications ($f$), the maximum number of iterations for the GA ($G$) and the solution pool size ($M$).

2. Generate a random pool of $M$ simple substitution cipher keys for position $j$ and calculate the cost for each using unigram statistics only. Call this pool of solutions $P$CURR.

3. For iteration $i$ ($i$= 1, ..., $G$) do:

   a) If $i$ mod $k'0$ send the best key from $P$CURR to each of the neighbouring GA's (i.e., the GA's solving for positions $j$ 1 and $j+1$). Also receive the best keys from each of these GA's.

   b) Select $M$ =2 pairs of solutions from $P$CURR to be the parents of the new generation. The selection should be biased towards the most fit of the current generation (i.e., the keys in $P$CURR).

   c) Mate using each pair of parents with the algorithm given above. This produces $M$ children that become the new generation (i.e., the solutions of $P$NEW).

d) Mutate each of the children in $P$NEW using the same swapping procedure as described in the attack on the simple substitution cipher.

e) Calculate the cost of each of the children in $P$NEW using the neighbouring keys obtained in Step 3a and Equation 3.1.

f) Select the $M$ best keys from the two pools $P$CURR and $P$NEW. Replace the current solutions in $P$CURR with these solutions.

4. Output the best key from $P$CURR.

Experimental results obtained from this algorithm are now given.

## IV. Results

It is clear that the parallel implementation of the attack will perform much more efficiently than a serial version since the parallel attack is solving each key of the polyalphabetic cipher simultaneously. The overhead of communication between the parallel processors is minimal leading to an attack of the polyalphabetic substitution cipher which would be expected to complete in roughly the same time as a similar attack on a monoalphabetic substitution cipher.

In this section results based on the amount of ciphertext provided to the attack are given. The attack was implemented with a polyalphabetic substitution cipher with a block size of three. The attack was run 100 times for each of 200, 400, 600, 800 and 1000 known ciphertext characters per key. The average results for the polyalphabetic substitution cipher are given in Fig. 5.

These results indicate that the parallel genetic algorithm is an extremely powerful technique for attacks on polyalphabetic substitution ciphers. It could be surmised from the experimental results given above that the attack could be used on polyalphabetic ciphers with very large periods provided that sufficient ciphertext and a parallel machine with sufficient nodes to implement the attack are available to the cryptanalyst.

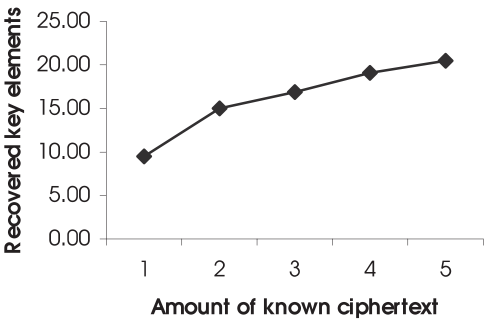

Fig. 5. Known ciphertext versus percent recovered (key and message).

## References

[1] J.Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan, 1975.

[2] D.E. Goldberg. Genetic Algorithms in Search. Optimization and Machine Learning. Addison Wesely, Reading, Massechusetts, 1989.

[3] K.A. DeJong. An Analysys of the Behavious of a Class of Genetic Adaptive Systems. University of Michigan Press, Ann Arbor, Michigan, 1975. Doctoral Disertation.

[4] R.Spillman, M.Janssen, B.Nelson, M.Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. Cryptologia, January 1993.

[5] A. Clark and E. Dawson. A parallel genetic algorithm for cryptanalysis of the polyalphabetic substitution cipher. *Cryptologia*, April 1997.