Chapter 6

TABU SEARCH FOR CRYPTANALYSIS

6.1 Introduction

Cryptanalysis of classical ciphers is the most popular cryptological application for meta-heuristic search investigations. The reasons for this are as follows. The basic concepts of substitution and transposition are still widely used today and so these ciphers form simple test beds for exploratory research. Problems of varying difficulty can be created say by altering the key size. With these objectives in mind and it presents in this Chapter experiments on cryptanalysis of SDES and modified versions of DES with 16, 32 and 48 bit keys. Moreover, cryptanalysis of simple ciphers is the best way towards a concrete understanding of cryptographic technologies, including the best way of getting exposed to the weakness of ciphers. Different experiments with and without SBOXes, and varying number of key bits have been conducted to study the attack of DES.

**Symmetric Cipher Model:** A symmetric encryption scheme has five ingredients shown in figure 6.1

![Figure 6.1: Simplified Model of Conventional Encryption](image)

**Plaintext:** This is the original readable intelligible message is fed into the algorithm as input [71][45-52].
Encryption Algorithm: The Encryption algorithm performs various substitutions and transformations on the plaintext.

Secret key: The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.

Ciphertext: This is the unreadable message or scrambled message produced as output from encryption algorithm [42]. It is completely depends on the plaintext and the secret key [36][37]. For example a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data, and as it stands, in unintelligible or unreadable.

Decryption algorithm: It is very essentially the encryption algorithm must run in reverse [71]. It takes the ciphertext or unreadable message and the secret key and produces the original plaintext [39][40][41].

There are two very important requirements for secure use of conventional encryption:

We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plain text that produced each ciphertext.

Sender and receiver must have obtained copied of the secret key in a secure fashion and must keep the key securely. If someone can discover the key and knows the algorithm, all communication using this key is readable.
We assume that it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decrypt algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 6.2: Model of Conventional Cryptosystem

A source produces a message in plaintext, \( X = [x_1, x_2, \ldots, x_M] \). The \( M \) elements of \( X \) are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet \( \{0, 1\} \) is typically used. For encryption, a key of the form \( K = [K_1, K_2, \ldots, K_J] \) is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.
With the message $X$ and the encryption key $K$ as input, the encryption algorithm forms the ciphertext $Y = [Y_1, Y_2, ..., Y_N]$. We can write this as $Y = E(K, X)$.

This notation indicates that $Y$ is produced by using encryption algorithm $E$ as a function of the plaintext $X$, with the specific function determined by the value of the key $K$. The intended receiver, in possession of the key, is able to invert the transformation: $X = D(K, Y)$.

An opponent, observing $Y$ but not having access to $K$ or $X$, may attempt to recover $X$ or $K$ or both $X$ and $K$. It is assumed that the opponent knows the encryption ($E$) and decryption ($D$) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover $X$ by generating a plaintext $X$. Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover $K$ by generating an $X$.

> Cryptography

Cryptographic systems are characterized along three important independent dimensions:

**The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations are reversible). Most systems, referred to as *product systems*, involve multiple stages of substitutions and transpositions.

**The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.

**The way in which the plaintext is processed.** A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.


**Cryptanalysis**

Typically, the objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis**: Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.
- **Brute-force attack**: The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

### 6.2 Tabu Search

The tabu search [58] prevents the search from returning to a previously explored region of the solution space too quickly. This is achieved by retaining a list of possible solutions that have been previously encountered. These solutions are called ‘tabu’ hence the name of the technique [65]. The size of the tabu list influences the performance of the algorithm. Tabu search is similar to simulated annealing. So consider any two randomly selected key elements are changed to generate applicant solutions. In each iteration, the best new key formed replaces the worst existing one in the tabu list.

To start with a random key which is also added to the tabu list and name it as current key and find out the fitness function of this key. In the original tabu algorithm, there are no specifications regarding how to perturb a particular solution. In this implementation and randomly perturb random number of bits in the current key to form a set of candidate keys. For this, a random number, n is generated. This number indicates the number of candidate solutions that are to be generated. To generate each of these candidate solutions and complement one bit of the current key. The bit to be complemented is also randomly selected.

Once this set of candidate keys is formed and calculate the fitness of each candidate key. To select the best key out of this set of candidate keys. To check whether this
best key is present in the tabu list. If not, this best key is added to the tabu list and this best key becomes current key. To perform the same steps as above with this key.

If the best key is in the tabu list and aspiration criterion is not satisfied (shortly describe the aspiration criterion), then look for the next best key in the set of candidate keys. This procedure is continued till a suitable candidate for the next current key is found or the set of candidate keys is exhausted. If the set of candidate keys is exhausted and do not change the current key and perform different random perturbation with the same key.

To keep track of the best key obtained so far, so that we can return this key at the end of the algorithm. The algorithm may be terminated by a limit on the number of iterations or if the correct key is found i.e. if fitness of a key is less than some very low threshold (about 0.09).

Tabu list size is the only one parameter it can manipulate in tabu search. The larger size of tabu list causes a lot of memory overhead. A very high value may even lead to segmentation faults. With smaller size of the tabu list, the list gets full quickly; hence it cannot continue the run for too long. If the old keys are deleted to make room for new keys and it may re-search the same search space destroying the whole purpose of tabu list. Thus, tabu list size is one critical parameter to choose.

Tabu search performed consistently better than other heuristics. The average time taken by tabu search to find the correct key was about 27 seconds. It took on an average 344 decryptions for finding the correct key. It is significantly better than 512 (average) decryptions in brute force attack. Note that, in this case, brute force attack is very simple and quite efficient too. Tabu search was consistent in the sense that it always provided the answer within the range of 20-40 seconds.

### 6.2.1 Tabu Search Template

Michel Gendreau at el. have [128] dealt with general template for tabu search, integrating the elements to minimize a function $f(S)$ (sometimes known as an objective or evaluation...
over some domain and we apply the so-called "best improvement" version of tabu search, i.e. the version in which one chooses at each iteration the best available move (even if this results in an increase in the function $f(S)$). This is the most commonly used version of tabu search.

**Notation:**

- $S$ the current solution,
- $S^*$ the best-known solution,
- $f^*$ value of $S^*$,
- $N(S)$ the neighborhood of $S$,
- $\hat{N}(S)$ the “admissible” subset of $N(S)$ (i.e. non-tabu or allowed by aspiration),
- $T$ tabu list.

**Initialization:**

Choose (construct) an initial solution $S_0$.

Set $S := S_0$, $f^* := f(S_0)$, $S^* := S_0$, $T := \emptyset$.

**Search:**

While termination criterion not satisfied do

- Select $S$ in $\text{argmin}[f(S')]$;
- $S' \in \hat{N}(S)$
  - if $f(S) < f^*$, then set $f^* := f(S)$, $S^* := S$;
  - record tabu for the current move in $T$ (delete oldest entry if necessary);

Endwhile.

In this algorithm, $\text{argmin}$ returns the subset of solutions in $\hat{N}(S)$ that minimizes $f$.

Generally, one must evaluate the objective function for every element of the
neighborhood \( N(S) \) of the current solution. This can be extremely expensive from a computational standpoint. In probabilistic tabu search, only a random sample \( N^\wedge(S) \) of \( N(S) \) is considered, thus significantly reducing the computational overhead. Another attractive feature is that the added randomness can act as an anti-cycling mechanism. This allows one to use shorter tabu fists than would be necessary if a full exploration of the neighborhood was performed. One the negative side, it is possible to miss excellent solutions. It is also possible to probabilistically select when to apply tabu criteria. Another way to control the number of moves examined is by means of candidate list strategies, which provide more strategic ways of generating a useful subset \( N^\wedge(S) \) of \( iV(S) \). In fact, the probabilistic approach can be considered to be one instance of a candidate list strategy, and may also be used to modify such a strategy. Failure to adequately address the issues involved in creating effective candidate lists is one of the more conspicuous shortcomings that differentiates a naive tabu search implementation from one that is more solidly grounded. Relevant designs for candidate list strategies are discussed in Glover and Laguna (1997).

Fatos Xhafa at el. dealt with tabu search (TS) algorithm for the problem of batch job scheduling on computational grids and define it as a bi-objective optimization problem, consisting of the minimization of the makespan and flowtime. TS is validated versus three other algorithms in the literature for a classical benchmark. We additionally consider some more realistic benchmarks with larger size instances in static and dynamic environments. Observed that that TS clearly outperforms the compared algorithms[101].

Fermín Alfredo at el. develop a tabu search algorithm to solve VRP_SPD. This algorithm uses three types of movements to obtain inter-route adjacent solutions. They are the relocation, interchange and crossover movements. A 2-opt procedure is used to obtain alternative intra-route solutions. Four types of neighbourhoods implemented, three of them defined by the use of each of the single inter-route movements and the fourth by using a combination of these movements[102]. Two different search strategies were implemented for selecting the next movement, first admissible movement and best admissible movement. Intensification and
diversification of the search were achieved through frequency penalization. Observed that TS procedure represents an improvement over former heuristics developed for the same problem [102].

6.3 Flowchart of a Standard Tabu Search Algorithm

The flow of operation using Tabu Search Algorithm on ciphertext is shown in the Figure 6.3. The tabu search prevents the search from returning to a previously explored region of the solution space too quickly. This is achieved by retaining a list of possible solutions that have been previously encountered.

![Flowchart of a Standard Tabu Search Algorithm](image)

Figure 6.3: Flow diagram of Tabu Search method

6.4 Experimental Setup and Results

Experiments are carried out to outline the effectiveness of Tabu Search (TS). The Algorithm of Tabu Search (TS) is coded in MATLAB 7 and tested on more than 100 benchmark data sets adapted. Among the unigrams, bigrams and trigrams, Unigram is
more useful and the benefit of trigram over digram is small. Table 6.1 shows the tabu search experimental results. The graphical representation for the values of the same is shown in Figure 6.4.

Table 6.1: Tabu Search Results

<table>
<thead>
<tr>
<th>Amount of Ciphertext (Character)</th>
<th>Tabu Search</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (Minute)</td>
<td>Number of bits matched in the Key</td>
</tr>
<tr>
<td>100</td>
<td>7.8</td>
<td>6</td>
</tr>
<tr>
<td>200</td>
<td>6.6</td>
<td>5</td>
</tr>
<tr>
<td>300</td>
<td>8.4</td>
<td>7</td>
</tr>
<tr>
<td>400</td>
<td>8.6</td>
<td>7</td>
</tr>
<tr>
<td>500</td>
<td>11.4</td>
<td>8</td>
</tr>
<tr>
<td>600</td>
<td>10.1</td>
<td>8</td>
</tr>
<tr>
<td>700</td>
<td>7.9</td>
<td>7</td>
</tr>
<tr>
<td>800</td>
<td>4.4</td>
<td>5</td>
</tr>
<tr>
<td>900</td>
<td>8.5</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>11.9</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 6.4: Tabu Search results
6.5 Concluding Remarks

This Chapter has demonstrated the efficacy of heuristic optimization techniques for the cryptanalysis of SDES and modified versions of DES. The various heuristic algorithms used for cryptanalysis are surveyed in this Chapter. To demonstrate the applicability of these algorithms to the problem of cryptanalysis, use these algorithms to candidate block and stream ciphers. Genetic algorithm is derived from the principles of evolutionary computation. In later Chapters the performance of these algorithms is evaluated and compared. Tabu Search is popular heuristic for both discrete and continues problems. as hardness flexibility is achieved using the Tabu Search algorithm. It achieved the amount of Ciphertext is 1000, time taken 11.9 and bits matched 9.

The results and test parameters of this method has been published in “Cryptanalysis of Simplified-Data Encryption Standard Using Tabu Search Method”, Published at ICIP 10th to 12th August 2012, CCIS 292, pp. 561-568, © Springer-Verlag Berlin Heidelberg 2012

The next chapter in this thesis concentrates on Experimental Analysis for Cryptanalysis Data Storage.