Automated Cryptanalysis

Otokar Grošek
Slovak University of Technology, Slovakia

Pavol Zajac
Slovak University of Technology, Slovakia

INTRODUCTION

Classical ciphers are used to encrypt plaintext messages written in a natural language in such a way that they are readable for sender or intended recipient only. Many classical ciphers can be broken by brute-force search through the key-space. One of the pertinent problems arising in automated cryptanalysis is the plaintext recognition. A computer should be able to decide which of many possible decrypts are meaningful. This can be accomplished by means of a text scoring function, based, e.g. on *n*-grams or other text statistics. A scoring function can also be used in conjunction with AI methods to speedup cryptanalysis.

BACKGROUND

Language recognition is a field of artificial intelligence studying how to employ computers to recognize language of a text. This is a simple task when we have enough amount of text with accents since they characterize used language with very high accuracy. Nowadays there are plenty of toolkits which automatically check/correct often both spelling and grammatical mistakes and errors. In connection with this we recall also the NIST Language Recognition Evaluation (LRE-05, LRE-07) as a part of an ongoing series of evaluations of language recognition technology. McMahon & Smith (1998) present an overview of natural language processing techniques based on statistical models.

We recall some basic notions from cryptography (see the article automated cryptanalysis of classical ciphers for more details). There is a reversible encryption rule (algorithm) how to transform plaintext to the ciphertext, and vice-versa. These algorithms depend on a secret parameter K called the key. The set of possible keys $\mathcal{K}$ is called the key-space. Input and output of these algorithms is a string of letters from plaintext, or ciphertext alphabet respectively. Both, sender as well as receiver, uses the same secret key, and the same encryption and decryption algorithms.

Cryptanalysis is a process of key recovery, or plaintext recovery without the knowledge of the key. In both cases we need a plaintext recognition subroutine which evaluates (with some probability) every candidate substring, whether it is a valid plaintext or not. Such automated text recognition requires an adequate model of a used language.

PLAINTEXT RECOGNITION FOR AUTOMATED CRYPTANALYSIS

In the process of automated cryptanalysis we decrypt the ciphertext with many possible keys to obtain candidate plaintexts. Most of the candidates are incorrect, having no meaning in a natural language. On the other hand, even the correct plaintext can be hard to recognize and with the wrong recognition routine can be missed altogether.

The basic type of algorithm suitable for automated cryptanalysis is a brute force attack. This attack is only feasible when key-space is searchable on computational resources available to an attacker. The average time needed to verify a candidate strongly influences the size of searchable key-space. Thus, the plaintext recognition is the most critical part of the algorithm from the performance point of view. On the other hand, only the most complex algorithms achieve really high accuracy of the plaintext recognition. Thus the complexity and accuracy of plaintext recognition algorithms must be carefully balanced.

A generic brute force algorithm with plaintext recognition can be described by the pseudo-code in Exhibit A.

Exhibit A.

INPUT: ciphertext string $Y = y_0 y_1 y_2 \ldots y_n$
OUTPUT: ordered sequence S of possible plaintexts with their scores

1. **Let** $S = \{\ \}$
2. **For each** key $K \in \mathcal{K}$ **do**
   - **2.1. Let** $X = d_K(Y)$ be a candidate plaintext.
   - **2.2.** Compute **negative test predicate** *filter(X).* **If** predicate is **true**, continue with step 2.
   - **2.3.** Compute **fast scoring function** *fastScore(X).* **If** *fastScore(X)* < *LIMITF*, continue with step 2.
   - **2.4.** Compute **precise scoring function** *score(X).* **If** *score(X)* < *LIMIT*, continue with step 2.
   - **2.5. Let** $S = S \cup \{<score(X), X>\}$
3. **Sort** S by key *score(X)* descending.
4. **Return** S.

Table 1. Performance of the three-layer decryption of a table-transposition cipher using a brute-force search. First filter was negative predicate-based, removing all decrypts with first 4 letters not forming a valid n-gram (about 90 % of texts were removed). Score was then computed as the count of valid tetragrams in the whole text. If this count was lower then given threshold (12), then the text was removed in the score-based filter. Finally, remaining texts were scored using the dictionary words.

Key-space Size	Negative filter	Score-based filter	Remaining texts	Total time [s]
9!	89.11%	10.82%	254	1.2
10!	89.15%	10.78%	2903	5.8
11!	88.08%	8.92%	239501	341
12!	90.10%	9.85%	1193512	746

Algorithm integrates the three layers of plaintext recognition, namely *negative test predicate, fast scoring function* and *precise scoring function,* as a three-layer filter. The final scoring function is also used to sort the outputs. First filter should be very fast, with very low error probability. Fast score should be easy to compute, but it is not required to precisely identify the correct plaintext. Correct plaintext recognition is the role of precise scoring function. In the algorithm, the best score is the highest one. If the score is computed in the opposite meaning, the algorithm must be rewritten accordingly.

In some cases, we can integrate a fast scoring function within the negative test or with the precise scoring, leading to two-layer filters, as in (Zajac, 2006a). It is also possible to use even more steps of predicate-based and score-based filtering, respectively. However, experiments show that the proposed architecture of three-layers is the most flexible, and more layers can even lead to performance decrease. Experimental results are shown in Table 1.

Negative Filtering

The goal of the *negative test predicate* is to identify candidate texts that are NOT plaintext (with very high probability, ideally with certainty). People can clearly recognize the wrong text just by looking at it. It is in the area of artificial intelligence to implement this ability in computers. However, most nowadays AI methods (e.g. neural networks) seem to be too slow, to be applicable in this stage of a brute-force algorithm, as every text must be evaluated with this predicate.

Most of the methods for fast negative text filtering are based on prohibited *n*-grams. As an *n*-gram