CS-GY 6903: Modern Cryptography

Professor: Giovanni Di Crescenzo

Project 1:
Breaking Polyalphabetic ciphers

Santiago Torres-Arias
NYU ID: N14553751

Lucas Mladek
NYU ID: N1xxxxxx

October 14, 2014

Contents

A Introduction

Polyalphabetic ciphers have been around in humanity for hundreds of years now. And, although their use is no longer recommended for real cryptography, its use is still widespread among cryptography amateurs and cryptography challenges[1].

It is known that polyalphabetic ciphers are still subject to frequency analysis, and known plaintext attacks. The focus of this writeup is to introduce techniques that leverage this knowledge to break these ciphers in the least possible time.

B Understanding polyalphabetic ciphers

B.1 The anatomy of the $j(i)$ function

We can try to decompose the $j(i)$ function by using calculus concepts. The following equation provides a simplified version of the possible variants that $j(i)$ can have:

$$j(i) = \left(\frac{xi^y}{zj^k} + m \right) \mod t \tag{1}$$

In this equation, we adopt the following nomenclature:

- i is the index
- x is a define a "decimating coefficient" for i.
- y defines a "decimating exponential" factor for the index.
- function zj^k defines a "stretching factor"
- m is a "start offset", and it can be described as shifting the key by m bins.
- mod t guarantees that the values selected are inside the generated key and t is the length of the key.

The relationship between the upper and lower factors define how this function behaves. In general, we can consider to be three variants: periodic, decimating and stretching. We will define these concepts next.

B.2 Key stretching and key-decimating

When the factor above (xi^y) is polinomially bigger than the factor below (zj^k), then we can assume that we have a key-decimating function. Key decimating functions make the key essentially "smaller" in a sense that some values are skipped and hence lost (depending on the periodicity, we will explain this later).

In the other hand, when values from the upper monomial are smaller than the lower one, we have what's called a key-stretching function. When this happens, we see that – due to integer

[1] A really popular cipher challenge includes two polyalphabetic ciphers http://simonsingh.net/cryptography/cipher-challenge/

math – some values are repeated contiguously. For example, imagine that we have the following $j(i)$:

$$j(i) = \frac{i}{4} \mod 30 \tag{2}$$

In this case, the resulting function can be considered stretching because values for i below 4 will all point to 0, which is the first element of the key.

The nature of this equation has a direct effect on what we will call the key's periodicity. We will explain this in the next subsection.

B.3 Periodicity

Periodicity of the key (and $j(i)$) can be understood as "the number of values for i before the key-selecting sequence repeats". Understanding this can allow us for an easy definition of a break function.

Regular periodic functions can be defined as those functions in which the lower factor is a multiple of t. When this happens, the periodicity of $j(i)$ also falls in a multiple of t. For our breaking scheme, we considered the function $j(i)$ to be a regular periodic function since it was the easiest to analyze. However, in Appendix B, we describe methods that could have been used to tackle the non-regular periodic family of functions.

B.4 Why periodicity matters and $j(i)$ not as much

If we consider a function to be regular periodic, then we can assume that the set of keys $a...z^t$ under a specific $j(i)$ is only a different permutation of another set of keys under a different $j(i)$. Knowing this, we can assume that, when a function is regular periodic, obtaining a proposed key for $j(i) = i \mod (t)$ is a safe bet. We will use this assumption in our breaking mechanisms, as it simplifies our analysis greatly.

C Breaking dictionary 1

In order to design the best-fastest mechanism to break the defined cipher, we first need to understand the nature of the existing dictionaries. In this case we know that the existing plaintexts falls in a range of 150 different plaintexts so we decided to analyse them as words.

C.1 Minimum prefix: plaintexts as words

Since the set of possible messages from Dictionary one is the same as the number of entries a dictionary 1, cryptanalysis is really easy. We could consider that the messages from dictionary one are a single, long word encrypted and sent through the wire.

To understand this, we build a python script called "minimum_prefix.py", that analyses the words in a dictionary to identify the minimum amount of characters needed to differentiate ciphertext from each other. The result of this script threw a value of 9, and we used it as part of our dictionary-header definition; this script can be found in the sources and is also described in Appendix A.

C.2 Defining our oracle

We can obtain a possible key by substracting a piece of plaintext from the ciphertext:

$$p_{[i]} + k_{[x]} = c_{[i]} \rightarrow k_{[x]} = c_{[i]} - p_{[i]} \tag{3}$$

Now, how do we know our key is correct?

C.3 Breaking dictionary 1: our attack

Breaking dictionary 1 is easy: we only need to apply the prefix plaintext to the provided ciphertext and compare the resulting key to another portion of the plaintext. The only catch is that we need to identify a region that uses the same range of the key. Since we assumed $j(i)$ to be regular periodic, then we can assume that it is a place where period starts and verify if the result matches correct plaintext of the same dictionary entry in that location.

The code we wrote for this takes less than a second to verify and dismiss all possible candidates for the plaintext.

D Breaking dictionary 2

D.1 Case study: Rosignol Cipher - The man with the Iron mask

https://en.wikipedia.org/wiki/Great_Cipher

D.2 Case Study: Breaking Indian Ciphers

http://shodhganga.inflibnet.ac.in/bitstream/10603/2193/13/13_chapter%205.pdf

D.3 Breaking polyalphabetic ciphers using triads(or trillables)

D.3.1 The size of the set

D.3.2 Probabilistic results

E Results

F Conclusion

Appendix A: Statistical analysis scripts and code