CS-GY 6903: Modern Cryptography

Professor: Giovanni Di Crescenzo

Project 1:
Breaking Polyalphabetic Ciphers

Santiago Torres-Arias
NYU ID: N14553751

Lucas Mladek
NYU ID: N13247152

October 17, 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.

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. (we assume y = 1)
- 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.
- we assume $\frac{x}{z}$ is a multiple of t, this makes it periodic on t.

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.

B.2 Key stretching and key-decimating

When the factor above (xi^y) is polinomially bigger than the factor below (z), 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).

On 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 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.

For key-stretching functions, we will be unable to decrypt the ciphertext.

For key-decimating functions, we will be able to decrypt the ciphertext if $\frac{x}{z}$ is a factor of t and therefore periodic on t.

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

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.

We will define 'regular periodic functions' as those functions in which $\frac{x}{z}$ 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, we could break Non-regular periodic functions by finding the same subset of the key in different places along the ciphertext. This is a regular step in breaking polyalphabetic cihpers if you are not given the key length. Since we were given the key length we did not take this approach.

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.

B.5 Frequency analysis on polyalphabetic ciphers

In most cases frequency analysis breaks polyalphabetic cihpers. However, for this project the cihper text we are able to sample is too small to perform frequency analysis on.

Kasiski Examination is another technique that would work and is similar to a frequency analysis but also requires us to be able to sample a greater amount of ciphertext.

C Breaking dictionary 1

In order to design the most efficient 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.

This is known as a 'known plaintext attack'. The dictionaries given were a small enough set to be analyzed using the methods outlined in this section.

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 plaintext from each other. The result of this script output a value of 9, and we used it as part of our dictionary-header definition; this script can be found in the sources.

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.2 Defining our Oracle and 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.

We used a few 'speed-ups' in our code, for example declaring the entire dictionary as a constant in C, alphabetizing the the dictionary, and using a 'trampolining' function to jump the the first letter of the alphabetized potential plaintext dictionary.

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

For dictionary 2 we can't use the same exact method as dictionary 1. So we researched different possible methods and case studies that were relevant to our attack. Before we explain our attack we will give a brief explanation of one such case study that help formulate our attack strategy.

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

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

The Rosignol's were a family of French cryptographers in the 17th century. The most famous of the family was Antoine Rossignol and his son Bonaventure who in 1626 developed a cryptographic code so strong it baffled cryptanalysts for centuries (wasn't decoded until 1893). This cipher used numbers to represent syllables rather than individual characters (the common approach at that time). The technical nature of the cipher includes a small 587 set of numbers that represent syllables and by itself is still subject to frequency analysis attacks.

In addition, this Cipher Schema seemed to pose a decryption/solution to solve the mystery of the man in the iron mask (infamous french prisoner at the time).

This inspired the approach we took to cracking dictionary 2. We focused on tuples of characters (like syllables but not quite) and their possible combinations in order to determine partials of words, or concatentations of 2 different words.

D.2 Breaking polyalphabetic ciphers using N-character tuples (trillables)

D.2.1 The size of the set

The size of the set is important to note for breaking dictionary 2 using tuples. Set groups as we make tuples bigger grow, but the probability of them appearing in the plaintext decreases. We found 3 tuples to be of an optimal size for our purposes and thus have dubbed them triads or trillables.

D.2.2 Probabilistic results

Every time we decrypt a 3-character tuple(trillable) there is a probability(of $\approx .1907$) that it belongs to the subset of all possible trillables.

In addition as the number of trials(in our case each trial is a decryption attempt) increases, the probability that all of the trillables fall into the set of possible trillables decreases.

Theoretically there are possible collisions that don't make this a fully encompassing approach. However, this method is much faster so we are trading quality for performance in order to make our overall decryption as fast as possible.

D.3 Defining our Oracle and Attack

We perform the same attack as in dictionary 1, however our oracle has changed. We now use the 3-character tuples (trillables) approach as defined in section D.2.2

E Results

We made a script that randomly selects a $j(i)$ with our constraints listed above and encrypts a plaintext from either dictionary 1 or 2. We used the results of this script to test our decryption program.

We utilized benchmarking functions and unit testing in our code for efficiency and troubleshooting respectively.

We also implemented a stopwatch timer in order to stop decryption attempts if our program began to run our of time. However our decryption schema/program runs so quickly we never needed to utilize it.

We found that our program can decrypt the ciphertext generated by our script in an extremely short amount of time relative to the 2 minutes upper limit for the assignment.

Lastly, we found no collisions that break our attack schema although admittedly they could indeed exist and would not permit for our attack against dictionary 2.

F Conclusion

We researched and found different and creative ways for breaking polyalphabetic ciphers. Some of which were implementable but not within the given time period for the project. Others were infeasible due to the dictionary being relatively short. Our attacks greatly depended on the fact the the key length was given and the plaintext was given beforehand (known plaintext attack).

Both partners Santiago and Luke were involved in the following tasks: Outlining and Testing Attack Strategies, Software Engineering and Coding, Analysis of Results, Write-Up and Final Analysis.