IN, LASEC: Bachelor Project #1

Due on Spring 2016

Pr. Serge Vaudenay

Max Premi

Abstract

This Hill cipher is a polygraphic substitution cipher based on linear algebra, invented by Lester S. in 1929. Each letter is represented by a number modulo 26, from $A = 0$ to $Z = 25$. The algorithm breaks the plaintext into blocks of size d and then applies a matrix $d \times d$ to these blocks to yield ciphertext blocks. As it's a linear encryption, it can be simply broken with Know PlainText Attacks. The author takes the previous paper about a new Ciphertext-only Attacks on Cipher Hill, and try to improve it's complexity to get a better result that $O(d13^d)$.

The goal of this project is to actually study the algorithm to get the key matrix modulo 2 and then to improve the algorithm to get the key matrix modulo 26.

The project report is organized as follows: Section1 presents the Hill cipher and the work done in the previous report. In section2, the author studies the complexity and try to improve the algorithm to get the key matrix modulo 26 . Section 3 presents the possible enhancement of the FFT of algorithm 1. Experimental results and algorithm are presented at the end.

Contents

Introduction

The motivation of this project is, first and foremost, to improve the Linear Cipher only attack on the Hill cipher, by changing the recovering of the key modulo 26 and then see the possible algorithm to improve the FFT in recovery of matrix key modulo 2.

Indeed, it is known that a brute force attack can be done on the Hill cipher, as it is a Linear cipher, but to have a better complexity and less restrictive resources, improvement have been made.
It is now possible to get a matrix key with minimum length required on ciphertext of $n = 8.96d^2 - O(\log d)$.
This method has been then improved [2] using the divide-and-conquer technique, and eliminating repeated calculation while doing matrix multiplication, and have led to a ciphertext required length of $n = 8,96d^2$.
Eventually, using the Chinese Reminder Theorem [2], the length has been brought to $n = 12.5d^2$, and the complexity to $O(d13^d)$.
By this same Chinese Reminder theorem, it is believed that we can find the key matrix modulo 2 first and then recover the matrix modulo 26 with a lower complexity [1].
It is shown in the previous paper that this matrix modulo 2 can be found in $O(d2^d)$.

Let's briefly describe how this attack works:
Let's consider X a random vector constituted of d letters, we can pick a fixed vector $\lambda \in \{0,1\}^*$ and consider the dot-product $\lambda.X$ in $\mathbb{Z}/2\mathbb{Z}$.
Then with the aid of the bias(X)= $\varphi_X(\frac{2\pi}{p})$ in $\mathbb{Z}/26\mathbb{Z}$, we found correspondence between λ and μ (the last is the same vector but for the cipher text). It is needed to search $bias(\lambda.X)$ and we found $\mu = (K^T)^{-1} \times \lambda$.
Then with this formula and the approximation of all the vector μ, we get the vectors column of the key matrix in $Z/2\mathbb{Z}$.
An algorithm to reorder them with the correlation, to find the last one and first one easily, and then recursively find all the vectors in the correct order.
All this process is described by algorithm 1 in the Annexe, and is done in a time $O(d2^d)$

This project will present possible improvement of this algorithm to get a lower complexity than the one mentioned before, with the help of Sparse Fourier Transform. Then, a possible enhancement to get the key matrix in modulo 26 will be discussed, as the one presented in the previous paper runs in $O(8^{nd})$.

Key recovery modulo 26

So now that we have the key matrix in $\mathbb{Z}/2\mathbb{Z}$, we can have the plain text in $\mathbb{Z}/2\mathbb{Z}$ using the linearity of the cipher.
To get the key matrix in $\mathbb{Z}/26\mathbb{Z}$, we can use the Chinese Reminder Theorem, but we would get a complexity proportional to $O(13^d)$. In the previous paper, it was believed that it's possible to get the key matrix in $\mathbb{Z}/26\mathbb{Z}$ without considering $\mathbb{Z}/13\mathbb{Z}$.
First of all, we create a hash table using long text, and search mapping between segments of reference text and plain text modulo 2.
#(seg in reference) = len(reference text) - n +1, with n the segment size.
Indeed, if the following text is taken as an example: *thisisatest*, with n = 5, we get the following segment: $thisi, hisis, isisa, sisat, isate, sates, atest$ which is 7 segments $11 - 5 + 1 = 7$
It's the same idea for #(seg in plain) $= len(plaintext) - n + 1$, with n the segment size.
Let a, b be random segment of length n and X the plaintext. We use Rényi entropy, with the following formula:

$$H_\alpha(X) = \frac{1}{1-\alpha}\log_2(\sum_{i=1}^{n}\Pr(X=i)^\alpha)$$

When alpha has the value 2, the following result is obtained:

$$-\log_2(\sum_{i=1}^{n}\Pr(X=i)^2)$$

that gives us the probability that a segment equals another one as $\sum_{i=1}^{n}\Pr(X=i)^2 = \sum \Pr(a=b)^2$. Rényi entropy represent more generally the quantity of information in the probability of a random variable's collision.

Then we define good matching : segments are equals before and after modulo 2, and bad matching segment which are not equal but equal modulo 2.

For good matching, we have $E(\#goodmatching) = (\#segmentsinreference) \times (\#segmentinplaintext) \times 2^{-H_2(X)}$, as the number of good matching is actually the collision between segment in plaintext and segment in reference text multiplied by the rényi entropy of this segment (which represents the rate of collision for a given block X).
Then the same is done for $E(\#allmatching)$, the difference is that it must be taken into account that we are in $\mathbb{Z}/2\mathbb{Z}$:$E(\#goodmatching) = (\#segmentsinreference) \times (\#segmentinplaintext) \times 2^{-H_2(X \bmod 2)}$. And indeed it is understandable that if 2 words modulo 2 are equals, these words are not always equals modulo 26.

Now let's condisder $E(\#allmatching)$. As we only 2 values are possible it's easier than the previous, indeed $E(\#allmatching) = (\#segmentsinreference) \times (\#segmentinplaintext) \times 2^{-H_2(X \bmod 2)}$.
$H_2(X \bmod 2) = -\log_2(\sum_{i=0}^{1}\Pr(X=i)^2)$, where $\Pr(X \bmod 2 = i)$ declined in $\Pr(X \bmod 2 = 0)$ and $\Pr(X \bmod 2 = 1)$
From the experiment, we always get 0.5^n for X mod 2 so E(# all matching) is never supposed to be different than $(\#segmentsinreference) \times (\#segmentinplaintext) \times 0.5^n$.
Then to have an idea of the complexity, the ratio $\frac{E(\#goodmatchings)}{E(\#allmatchings)}$ is computed, to find a general expression to express the number of good matching in all matching:$\frac{1}{8^n}$
In the Experiment part, the computation of $E(\#allmatchings)$ are done again with the help of a Java program. To decrease the actual complexity, we need to increase the ratio of good matching as $E(\#allmatchings)$ can't be changed. So the only solution left is to try different assumptions and calculations for $E(\#goodmatchings)$.

The one that is interesting is to consider blocks of letters as being independent from each others, and look at the evolution of the ratio through the growing block size. This is done in Experiment 2.
We finally conclude that it's possible to have a correct ratio for large size block, but the actual algorithm depends too much on the blocksize, and it's therefore impossible to get a correct complexity with the found curve that looks more like. For example, $blocksize = 27$ gives ratio $\frac{1}{5}$ but still $\frac{1}{ratio^{blocksize}}$ is too high as $blocksize = 27$. Eventually, the following parts focus on other way to implement this recovery of key matrix modulo 26.

Study of Algorithm to get K_{26}

We are going to try to improve the complexity of algorithm 2 to find another complexity than $O(8^{nd})$ with n the segment size and $d \times d$ the matrix size.
We want to turn the problem in another way, meaning instead of looking at all possible matching and do all the decryption possible with d matching, try to found the number of good matching we need so that an

algorithm can find the key matrix by solving equations.
So the problem can be turned like this:find the number y of matching needed to have a set of linear equation of the first order, to find the matrix coefficient in $\mathbb{Z}/26\mathbb{Z}$.
To be clear, let's recall what we are given. We got cipher $Y_1, Y_2, ..., Y_n$ in $\mathbb{Z}/26\mathbb{Z}$ but also in $\mathbb{Z}/2\mathbb{Z}$, the matrix K_2 thanks to algorithm 1 , and from these two we get the plaintext in $\mathbb{Z}/2\mathbb{Z}$, $X_1, X_2, ..., X_n$. We now create a matrix K$_{26} of the same size and with element$x$_1, x_2, ..., x_{d^2} \in \mathbb{Z}^{13}$.

Study of Faster Fourier Transform for Algorithm 1

With a fast Fourier Transform (FFT) the complexity is $O(N \log N)$ for N the input size.
But generally, most of the coefficient of a FFT are small or equal to zero, meaning the output of the FFT is sparse. If a signal has a small number k of non-zero Fourier coefficients the output of the Fourier transform can be represented succinctly using only k coefficients. Hence, we can find Fourier Transform algorithm whose run time is sub-linear in the signal size n.
what we want is to enhance the possible FFT on a table called n_y which contains the number of times k where each cipher y appears. So it is a table containing numbers $\in \mathbb{N}$.
The actual probability that all vectors are different is:

$$\prod_{n=0}^{k-1} (26^{-d})(26^d - n)$$

If we suppose that they actually are 26^d ciphers.
But it's not the case, so from our experiment on independent block, we also computed the probability that a block equals another one. We just need to take this value for all the different blocksize, and then, as we assumed all the blocks where independent, put it at the power number of blocks.
Which gives us:

$$(1 - \Pr(a = b))^{\#ciphers}$$

With this formula the bigger the size of block is, the more ciphers we can get and are different for sure. You can see Experiment 3 to look the probability with some blocksize.

Simple and practical algorithm for sparse Fourier transform

This algorithm considers a complex vector x of length n.
It computes the k-sparse Fourier transform in $O(\sqrt{kn} \log^{3/2} n)$, if x is sparse then you find it in exactly $O(klog^2 n)$, but in general estimate x is approximately $O(\sqrt{nk})$
So this algorithm is better if the ratio $\frac{n}{k} \in [2 \times 10^3, 10^6]$, but it's clearly not the best one as recently found are supposed to find it in a lower complexity $(k \log(n))$.
But it can still be used and gives correct results. So we will assume all ciphers are different, and we get all non null component in n_y from the result in experiment 2, of size n, meaning n given ciphers. We find that the Fourier transform only get high picks at the extremities.
So if we consider that the middle are null, we can reduce the complexity.
For example, for blocksize = 10 and 6200 ciphertext , the probability that they are all different is $0.99999683297^{6200} = 0.9805$. So we assume they are all different, and find the following FFT for a vector of 100 one and 200 one: