IN 101: Bachelor Project #1

Due on Monday, January 1, 2012

Pr. Serge Vaudenay 10:30am

Max Premi

Contents

Introduction

The motivation of this project is first and foremost to improve the Linear attack on the Hill cipher , by changing the recovery of the key modulo 26 and then see the possible algorithm to improve the FFT.
Let's briefly recall how this attack works.
You get the plaintext into modulo 2 , then with the aid of vectors , and bias
bias(X)= $\varphi_X(\frac{2\pi}{p})$ in $\mathbb{Z}/26\mathbb{Z}$
With the fact that the plain text is of length d , we can write λ X that represent the plain text , where X is a vector column.
Then thanks to the bias , we found correspondence between λ and μ (which is the same vector but for the cipher text). Wa actually get $\mu = (K^T)^{-1} \times \lambda$
Then with this formula and the approximation of all the vector μ , we get the column of the key matrix.
Then we need to find the correct order in the key matrix , we use algorithm 1

Key recover 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 of $O(13^d)$. 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 you take the following text : *thisisatest* , with n = 5 , you get the following segment:
$thisi, hisis, isisa, sisat, isate, sates, atest$ which is 7 segments $11 - 5 + 1 = 7$
We get the same thing for #(str in plain) = len(plain text) - n +1 , with n the string size.
Then we define the good mathcings : segments are equals before and after modulo 2 , and bad matching segment are not equal but they are equal modulo 2.
We use Rnyi entropy to get the good matching and all matching as it find the collision , with the following formula :
$H_\alpha(X) = \frac{1}{1-\alpha} log_2(\sum_{i=1}^{n} Pr(X = i)^\alpha)$, then when alpha has the value 2 , we just get $-log_2(\sum_{i=1}^{n} Pr(X = i)^2)$ that gives us the probability that a segment equals another one.
For good matchings , we have E(# good matchings) = (# segments in reference) x (#segment in plaintext) x $2^{-H_2(X)}$, as the number of good matching is actually the collision between segment in plaintext and segment in reference text time the entropy of rnyi where two segments are the same.
Then you do the same for E(# all matchings) , the difference is that you do it this way : E(# good matchings) = (# segments in reference) x (#segment in plaintext) x $2^{-H_2(Xmod2)}$. And indeed you understand that if X modulo 2 are equals the X are not always equals.
The previous part assumed that $H_2(Xmod2)$ was equals to n , but i did these maths again and it seems good.
Then to have an idea of the complexity , you do the ratio $\frac{E(\#goodmatchings)}{E(\#allmatchings)}$, you generaly found $\frac{1}{8^n}$
In the following parts , I did the calculation again to check if it's really correct
But to have a better complexity , we need to increase this ratio : E(# all matchings) can't be changed so we can only try on E(# good matchings) , with different assumptions and calculations.

Experiment

from wiki :

Probabilite de la 1ieme lettre de l'aphabet 0.08167
Probabilite de la 2ieme lettre de l'aphabet 0.01492
Probabilite de la 3ieme lettre de l'aphabet 0.02782
Probabilite de la 4ieme lettre de l'aphabet 0.04253
Probabilite de la 5ieme lettre de l'aphabet 0.12702
Probabilite de la 6ieme lettre de l'aphabet 0.02228
Probabilite de la 7ieme lettre de l'aphabet 0.02015
Probabilite de la 8ieme lettre de l'aphabet 0.06094
Probabilite de la 9ieme lettre de l'aphabet 0.06966
Probabilite de la 10ieme lettre de l'aphabet 0.00153
Probabilite de la 11ieme lettre de l'aphabet 0.00772
Probabilite de la 12ieme lettre de l'aphabet 0.04025
Probabilite de la 13ieme lettre de l'aphabet 0.02406
Probabilite de la 14ieme lettre de l'aphabet 0.06749
Probabilite de la 15ieme lettre de l'aphabet 0.07507
Probabilite de la 16ieme lettre de l'aphabet 0.01929
Probabilite de la 17ieme lettre de l'aphabet 9.5E-4
Probabilite de la 18ieme lettre de l'aphabet 0.05987
Probabilite de la 19ieme lettre de l'aphabet 0.06327
Probabilite de la 20ieme lettre de l'aphabet 0.09056
Probabilite de la 21ieme lettre de l'aphabet 0.02758
Probabilite de la 22ieme lettre de l'aphabet 0.00978
Probabilite de la 23ieme lettre de l'aphabet 0.02361
Probabilite de la 24ieme lettre de l'aphabet 0.0015
Probabilite de la 25ieme lettre de l'aphabet 0.01974
Probabilite de la 26ieme lettre de l'aphabet 7.4E-4

proba sum = 0.9999999999999999
proba sum squared = 0.06549717159999999
proba sum squared 0 = 0.56832 0.32298762240000006
proba sum squared 1 = 0.43167999999999995 0.18634762239999997
Ration of good matching and all matchings=0.1285934407027314
Donc 7,77644

Another site :
Probabilite de la 1ieme lettre de l'aphabet 0.0808
Probabilite de la 2ieme lettre de l'aphabet 0.0167
Probabilite de la 3ieme lettre de l'aphabet 0.0318
Probabilite de la 4ieme lettre de l'aphabet 0.0399
Probabilite de la 5ieme lettre de l'aphabet 0.1256
Probabilite de la 6ieme lettre de l'aphabet 0.0217
Probabilite de la 7ieme lettre de l'aphabet 0.018
Probabilite de la 8ieme lettre de l'aphabet 0.0527

Probabilite de la 9ieme lettre de l'aphabet 0.0724
Probabilite de la 10ieme lettre de l'aphabet 0.0014
Probabilite de la 11ieme lettre de l'aphabet 0.0063
Probabilite de la 12ieme lettre de l'aphabet 0.0404
Probabilite de la 13ieme lettre de l'aphabet 0.026
Probabilite de la 14ieme lettre de l'aphabet 0.0738
Probabilite de la 15ieme lettre de l'aphabet 0.0747
Probabilite de la 16ieme lettre de l'aphabet 0.0191
Probabilite de la 17ieme lettre de l'aphabet 9.0E-4
Probabilite de la 18ieme lettre de l'aphabet 0.0642
Probabilite de la 19ieme lettre de l'aphabet 0.0659
Probabilite de la 20ieme lettre de l'aphabet 0.0915
Probabilite de la 21ieme lettre de l'aphabet 0.0279
Probabilite de la 22ieme lettre de l'aphabet 0.01
Probabilite de la 23ieme lettre de l'aphabet 0.0189
Probabilite de la 24ieme lettre de l'aphabet 0.0021
Probabilite de la 25ieme lettre de l'aphabet 0.0165
Probabilite de la 26ieme lettre de l'aphabet 7.0E-4

proba sum = 0.9999000000000001
proba sum squared = 0.06609151
proba sum squared 0 = 0.5657 0.32001649
proba sum squared 1 = 0.4342 0.18852964
Ration of good matching and all matchings=0.12996168115565054
Donc 7,69457

Enhancement good matching's ratio

This section will be to increase the ratio found which is actually $\frac{1}{8^n}$
To do so , I'll now consider instead of independent letters , independent blocks of letter. With the help of a Java programm , i'm doing an heuristic search over a very very long text , with different block size.
With the program , we clearly see that there is no way to improve it considering that they are independent.

Study of Faster Fourrier Transform for Algorithm 1

lololo

Algorithm

You hash a reference text.
You take the key matrix that you get from algorithm 1 , find plain text in $\mathbb{Z}/2\mathbb{Z}$, and create an array.
find the list of all matchings : find all pairs(seg,str) such that seg is a segment of plaintext modulo 2 and str $\in hash(seg)$ and save it in a list.
repeat
select d matching form list (you'll get a dxd key matrix)

for each of these matchings (seg_i, str_i)
extract $block_i$ from seg_i and str'_i from str_i,
then find $ciphertext_i$ such that K^{-1} x $ciphertext_i$ mod 2 = $block_i$
solve $ciphertext_i = K * str'_i$ for i=1 to d
compute $K^{-1}*$ciphertext
until it makes sense
number of iteration is $\frac{1}{ratio^{nd}} = 8^{nd}$

The following algorithm is to recover the key matrix in $\mathbb{Z}/2\mathbb{Z}$
Part1:

You require Ciphertext $Y_1, Y_2, ..., Y_n$
for all μ do compute $S_n(\mu) = \sum_{k=1}^{n} (-1)^{\mu.y} \times n_y$ where $n_y = \#\{k; Y_k = y\}$
endfor
set all μ to the d values of μ with largest $S_n(\mu) = bias(\mu.Y)$

Part2:

for all (i,i') do
compute $n_{00}(i, i') = \#\{k < n : (\mu_i.Y_k, \mu'_i.Y_{k+1}) = (0, 0)\}$ endfor
set (i_d, i_1) to the first pair with lowest n_{00}