IN, LASEC: Bachelor Project #1

Due on Spring 2016

Pr. Serge Vaudenay

Max Premi

Abstract

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 than $O(d13^d)$.

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:Section 1 presents the Hill cipher and the work done in the previous report. In Section 2, 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 algorithms are presented at the end.

Contents

1 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.

The 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 key matrix $d \times d$ to these blocks to yield ciphertext blocks.
In order to encrypt a message, the ciphertext is calculated as a matrix product of the key and plaintext:

$$Y = K \times X$$

And in order to decrypt, we simply use linear algebra properties and multiply the cipher by the inverse of the key matrix.

$$X = K^{-1} \times Y$$

As it's a linear encryption, it can be simply broken with Known PlainText Attacks.
Indeed, it is known that a brute force attack can be done on this cipher, by intercepting d^2 plaintext/ciphertext character pairs, however to have a better complexity and less restrictive resources, improvement have been made to this simple brute force attack.
The attack described previously can determine the secret key almost uniquely if the length of ciphertext is at least $n = 1.27d^2$[1].
Later, it has been proved that it is possible to get the key matrix K, with minimum length of ciphertext $n = 8.96d^2 - O(\log d)$ [1].
This method has been then enhanced [1] 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 Remainder Theorem [1], the length has been brought to $n = 12.5d^2$, and the complexity to $O(d13^d)$.
By this same Chinese Remainder 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 [2].
It is shown in the previous paper that this matrix modulo 2 can be found in $O(d2^d)$, and the key matrix modulo 26 in $O(8^{nd})$.

Let's briefly describe how this attack works:
If we consider X a random vector constituted of d letters, then we can pick a fixed vector $\lambda \in \{0,1\}^*$ and do the dot-product $\lambda \cdot X$ in $\mathbb{Z}/2\mathbb{Z}$.
The d vectors λ with largest non-trivial bias are obtained when $weight(\lambda) = 1$, and are used from now on for all computations. Then with the aid of the bias(X)= $\varphi_X(\frac{2\pi}{p})$ in $\mathbb{Z}/26\mathbb{Z}$, we find correspondence between λ and μ (the last is the same vector but for ciphertext). More precisely we find correspondence between bias($\lambda \cdot X$) and bias($\mu \cdot Y$) and get $\mu = (K^T)^{-1} \times \lambda$.
With this result and the approximation of all vectors μ via $S_n = \sum_{k=1}^{n} (-1)^{\mu \cdot Y_k}$, we recover the vectors column of the key matrix in $Z/2\mathbb{Z}$.
An algorithm to reorder them in function of the correlation is used to identify 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 Appendix, and is done in time $O(d2^d)$

This project will present possible improvements 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 $O(8^{nd})$.

2 Key recovery modulo 26

So now that we have the key matrix in $\mathbb{Z}/2\mathbb{Z}$, we can get the plain text in $\mathbb{Z}/2\mathbb{Z}$ using the linearity of the cipher.
To recover the key matrix in $\mathbb{Z}/26\mathbb{Z}$, we can use the Chinese Remainder 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.

Assumption 1. *All segments of length n are independent with the same distribution*

Theorem 1. *Rényi entropy of order α where $\alpha \geq 0$ and $\alpha \neq 1$:*
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 $\alpha = 2$, the following result is obtained:

$$H_2(X) = -\log_2(\sum_{i=1}^{n} \Pr(X = i)^2)$$

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

Then we define good matching:segments are equal 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 $\mathbb{Z}/2\mathbb{Z}$:$E(\#allmatching) = (\#segmentsinreference) \times (\#segmentinplaintext) \times 2^{-H_2(X \bmod 2)}$. And indeed it is understandable that if 2 words modulo 2 are equals, these words are not always equals modulo 26.

As we consider independent letters in segment, the probability to find two of the same segment with following ocurence of letters:$p(A) = 0.0808, p(B) = 0.0167$, ...etc:$\Pr(a = b) = ((0.0808)^2 + (0.0167)^2 + ...)^n = 0.06609^n$.
Then $H_2(X) = -log_2 0.06609^n$. The frequencies given in [8] are used as reference.

Now let's consider $E(\#allmatching)$. As we only 2 values are possible it's easier than the previous, indeed $E(\#allmatching) = (\#segmentsinreference) \times (\#segmentinplaintext) \times 2^{-H_2(X \bmod 2)}$.
$H_2(X \bmod 2) = -\log_2(\sum_{i=0}^{1} \Pr(X \bmod 2 = i)^2)$, 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 distribution of good matching and bad matching, the ratio $\frac{E(\#goodmatchings)}{E(\#allmatchings)}$ is computed: $\frac{E(\#goodmatchings)}{E(\#allmatchings)} = \frac{|segmentsinreference| \times |segmentinplaintext| \times 2^{-H_2(X)}}{|segmentsinreference| \times |segmentinplaintext| \times 2^{-H_2(X \bmod 2)}} = \frac{0.06609^n}{0.5^n} = \frac{1}{7.56^n} \cong \frac{1}{8}$

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 take a very long sample text, and count heuristically the number of good and all matching. We finally conclude that it's possible to have a correct ratio for large size block, but as the actual algorithm depends too much on the blocksize, it is therefore impossible to get a correct complexity. 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 key matrix recovery modulo 26.

3 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 find 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 $\mathbb{Z}/26\mathbb{Z}$.

3.1 Computation with conditional entropy

If the previous part is considered, it is simple to say that we can calculate entropy in another way. What one can do, is simply take the conditional entropy.

Theorem 2. *Conditional Rényi entropy of order α where $\alpha \geq 0$ and $\alpha \neq 1$:*

$$H_\alpha(X \mid X \bmod 2 = y) = \frac{1}{1-\alpha} \log_2(\sum_{i=1}^{n} \Pr(X = i \mid X \bmod 2 = y)^\alpha), y \in \{0, 1\}$$

For $\alpha = 2$, $-\log_2(\sum_{i=1}^{n} \Pr(X = i \mid Xmod2 = y)^2)$
In the case of single independent letter, the entropy declines into 2 possibilities, $H_2(X \mid X \bmod 2 = 0)$ and $H_2(X \mid X \bmod 2 = 1)$. The conditional probability is as following:

$$\Pr(X = i \mid X \bmod 2 = 0) = \frac{\Pr(X \bmod 2 = 0 \mid X = i) \Pr(X = i)}{\Pr(X \bmod 2 = 0)} =$$

$$\begin{cases} \frac{\Pr(X=i)}{\Pr(X \bmod 2=0)} & if \Pr(X \bmod 2 = 0|X = i) \neq 0 \\ 0 & else \end{cases}$$

and of course the same for 1.

Therefore, $\sum_{i=1}^{n} (\Pr(X = i \mid X \bmod 2 = 0))^2 = ((\frac{\Pr(X=A)}{\Pr(X \bmod 2=0})^2 + (\frac{\Pr(X=C)}{\Pr(X \bmod 2=0})^2 + ...) = 0.112$, where $p(A) = 0.0808, p(B) = 0.0318, ...$ we get the probability from [8] exactly like the previous section. And $\Pr(X = i \mid X \bmod 2 = 1) = 0.128$
We can now estimate the good matching by "class" of reduction modulo 2 Experiment 3 gives more details about the consideration of blocks instead of independent letters. We can now get the ratio $\frac{E(\#goodmatchings)}{E(\#allmatchings)}$ in class of modulo 2 reduction, so conditional Expected value with the following : $\frac{E(\#goodmatchings|X \bmod 2=y)}{E(\#allmatchings \bmod 2=y)}$ or more clearly $\frac{E(\#goodmatchings|Xinclassy)}{E(\#allmatchingsinclassy)}$. But in one class all the blocks are matching modulo 2 so it just gives $E(\#goodmatchings \mid Xinclassy) = \Pr(X = i \mid X \bmod 2 = y)$

Let's now try to resolve the number of equation needed so that we can resolve the key matrix modulo 26.

We get the cipher with the formula $K \times X = Y$, if we get n good matching , $K \times \begin{pmatrix} - \\ - \\ ... \\ a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} u \\ v \\ ... \\ - \\ - \\ - \\ - \end{pmatrix}$ where $u, v, ...$ represent the good matching and the $d - n$ others are not.
For this to work we need $d - n < n$ meaning $n > \frac{d}{2}$
If we call $\Pr(\#GoodMatching) = p^n$ and we get N matching , we get Np^n good matching.

4 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.
A general algorithm for computing the DFT must take time proportional to its output size N. However , in some cases, most of the Fourier coefficients of a signal are small or equal to zero, meaning, the output of the DFT is sparse.
For sparse signals, the n lower bound for the complexity of DFT no longer applies. 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 $\in \mathbb{N}$ of size $N = 2^d$.

4.1 Simple and practical algorithm for sparse Fourier transform

This algorithm considers a complex vector x of length l.
It computes the k-sparse Fourier transform in $O(\sqrt{kl} \log^{3/2} l)$, if x is sparse then find it takes exactly $O(klog^2 l)$, but in general estimate x is approximately $O(\sqrt{lk})$
So this algorithm is better if the ratio $\frac{l}{k} \in [2 \times 10^3, 10^6]$, but it's clearly not the best one as recently found are supposed to find it in a lower complexity ($k \log(l)$).

Now let's consider the input of this DFT to be n_y with size 2^d. This table contains integer whose the sum is equal to the number of cipher given. As a result, we can't say that the resulting DFT of this vector will be sparse or not.

4.2 Deterministic Sparse Fourier Approximation via Fooling Arithmetic Progressions

Here if we gave a threshold $\tau \in (0,1]$ and an oracle access to a function f, it outputs the τ-significant Fourier Coefficient. This is called SFT and runs in $\log(N), \frac{1}{\tau}$.
An oracle access to a function take as input x and return the $f(x)$ of the function f.
This algorithm is robust to random noise and local (meaning it runs in polynomial time)
It's based on partition of set by binary search, we have at the beginning 4 intervals, then testing for the first two if the norm of f Fourier Transform squared is equals to the set_i oracle output squared.
Meaning more explicitly : $f(J_i)^2 = \sum_{\alpha \in J_i} |f(\alpha)|^2$ If this pass, it will output yes, and we'll be able to continue the algorithm by replacing the J and insert the J_i
The heart of the code is actually to decide which intervals potentially contain a significant Fourier coefficient. Yes if weight on J, exceeds significant threshold τ, NO if J larger.
The threshold τ can be chosen, with the fact that a α is a $\tau - significant$ Fourier coefficient iff $|\hat{f}|^2 \geq \tau ||f||_2^2$ where $\hat{f} = \langle f, X_\alpha \rangle$ and $X_\alpha = e^{2\pi i \alpha x/N}$.
Considering the table n_y of size 2^d, we get $\tau ||f||_2^2 < \tau \times n^2$ as all the cipher are in there and there are n ciphers were $2^d < n$. So the semi-norm will be the smallest if they are all distributed equally in the "buckets". That gives approximately $\lfloor \frac{n}{2^d} \rfloor^2 * 2^d$ and the worst case where the semi norm is the bigger is when all are in one place meaning n^2.
Let's take the value from the example of the previous paper: 6200 ciphers and blocksize =10. There for the best case, i.e, all equally distributed each bucket contains $\lfloor \frac{1024}{6200} \rfloor^2 = 6$, we get $|\hat{f}|^2 \geq \tau ||f||_2^2 = 6144^2 \geq \tau 36864$, where 6024 is the biggest Fourier coefficient and all the others are 0 so $1 = \tau$ that will be a good result but the probability that it arrives is clearly too small to be considerate.
For the worst case we would get:$|\hat{f}|^2 \geq \tau ||f||_2^2 = \alpha \geq \tau 6200^2$ where α is one of the smallest coefficient but still bigger the order 10^1 The problem is that τ will need to be very big to match the previous inequality. So the complexity will depends on $\frac{1}{\tau}$ and it'll clearly be too important to be considered.

4.3 Nearly optimal Sparse Fourier Transform

We want here to compute the k-sparse approximation to the discrete Fourier transform of an 2^d-dimensional signal.
In the case where the input has at most k non-zero Fourier coefficient, we got $O(k.\log(2^d))$ time, else we have $O(k.\log(2^d).\log(\frac{2^d}{k}))$
The basis is still the same, if a signal has a small number k of non-zero Fourier coefficient, the output of this DFT can be represented succinctly using only k coefficient.
What is required, is that the input size n is a power of 2 which is complete in this case.
This algorithm has a better performance if and only if $k < O(\frac{2^d}{\log(2^d)}$. This won't be the case here and the formula for the k superior to this limit perform too poorly for worst case:$O(\sqrt{(}2^d k) \log^{\frac{3}{2}}(2^d))$

Experiment

Experiment 1:Probability of independent English letters

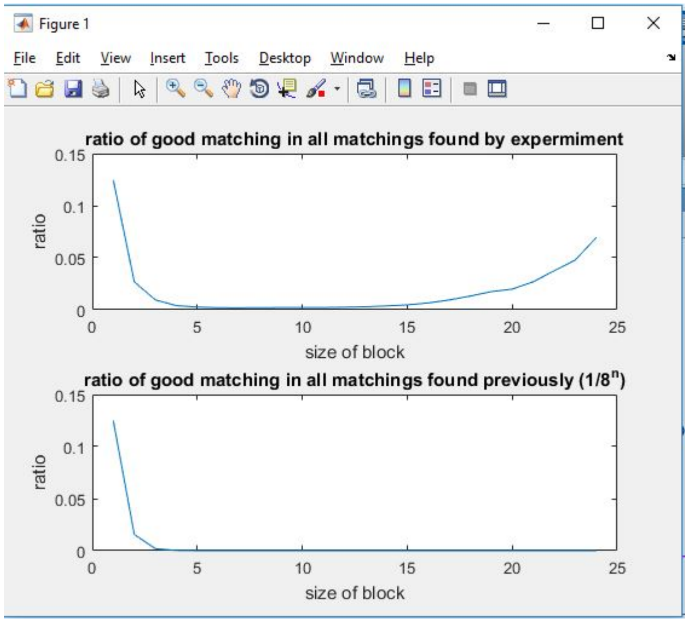
From the frequency letter given by Wikipédia, in english we got the following result :
Proba sum = 0.9999999999999999
Sum of probability squared = 0.06549717159999999, which corresponds to $(\sum_{i=0}^{25} \Pr(i = y)^2)^n, y \in \{alphabet\}$
Sum of probability that gives 0 modulo 2 squared = 0.32298762240000006 which corresponds to $(\sum_{i=0}^{25} \Pr(i = 0)^2)^n, i \in \{alphabet \bmod 2\}$
Sum of probability that gives 1 modulo 2 squared = 0.18634762239999997 which corresponds to $(\sum_{i=0}^{26} \Pr(i = 1)^2)^n, i \in \{alphabet \bmod 2\}$
Ration of good matching and all matching=0.1285934407027314^n
So $\frac{1}{7{,}77644^n}$.

Another site [8], with a total number of 100000 letters composed with texts from Edgar Allan Poe, Arthur Conan Doyle, and 4 articles from encyclopedia Encarta 95:

proba sum = 0.9999000000000001
sum of probability squared = 0.06609151 which corresponds to $(\sum_{i=0}^{25} \Pr(i = y)^2)^n, y \in \{alphabet\}$
sum of probability that gives 0 modulo 2 squared = 0.32001649 which corresponds to $(\sum_{i=0}^{25} \Pr(i = 0)^2)^n, i \in \{alphabet \bmod 2\}$
sum of probability that gives 1 modulo 2 squared = 0.18852964 which corresponds to $(\sum_{i=0}^{25} \Pr(i = 1)^2)^n, i \in \{alphabet \bmod 2\}$
Ration of good matching and all matching=0.12996168115565054^n
So $\frac{1}{7{,}69457^n}$.

Experiment 2:Probability considering blocks of size d

So calculation are done on a text of approximately 860000 characters to see the evolution of the ratio good matching/bad matching.
A program is ran to see the evolution for a block size between 1 and 25, and give the ratio, thanks to the probability that a block appears. It is completely heuristic as it's just counting the number of block that appears and do some manipulation with it. So the basic is to choose a block size, then it'll count every different blocks that appears modulo 26 and modulo 2. Then it'll compute the probability that a good matching happen with the following : $\sum_{X \in block} (\frac{\#X-1}{\#block-1})^2$
The exact same thing is done with X in modulo 2, to get the probability of all matching, and then we compute the ratio $\frac{E(\#goodmatchings)}{E(\#allmatchings)}$
With this, the evolution of the ratio in function of the block size looks like this:



So we can see that the ratio follow the $\frac{1}{8^n}$ until blocksize 15, then it goes up again to almost match 1/8 for blocksize 24. With the current algorithm and this size of block the number of iteration would be 8^d and as $d = 24$ is really large it's still not effective. Even if the ratio do not behave as previously thought, the complexity stay too high for reasonable blocksize between 8 and 14, and if the blocksize is increased to a certain point, the ratio is good but the complexity depends on a ratio power the blocksize, so it will not be good enough to be taken in account.

Experiment 3:Conditional Probability considering that we know X mod 2

In this experiment, we still consider the text from the previous part and sort block of letter in class of reduction modulo 2. For example the block *ab* will be in the class 01. From this sorting we can know which class has the best ratio of good matching. We know select the best class as following
The best class is not the class that have the best ratio of good matching over all matching, as we can have a class where there is only 3 matching and 2 are equals. So to avoid classes like this we just take the class with the highest number of good matching. To have an idea of the number of block, it is almost 847400.

blocksize	Best Class	Ratio of good matching	# block from this class	Bigger blocksize
1	0	0.119870	478261	478261
2	11	0.062238	122847	246310
3	110	0.033030	94719	151590
4	1110	0.02486952	24273	81141
5	10110	0.0098346	36098	48807
6	011110	0.0360992	3300	25434
7	1010001	0.00833618	10022	15094
8	10100010	0.0199979	6219	8132
9	100110101	0.03706977	3127	4544
10	1001101010	0.075711852	2183	2514
12	000110100010	0.1072417	654	784
14	10101110100010	0.332720389	215	287
16	1000010000100011	0.6810255	103	122

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 matching: 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.

```
1: repeat
2:    select d matching form list (you'll get a $d \times d$ key matrix)
3:    for each of these matchings $(seg_i,\ str_i)$ do
4:       extract $block_i$ from $seg_i$ and $str'_i$ from $str_i$,
5:       then find $ciphertext_i$ such that $K^{-1}$ x $ciphertext_i$ mod 2 $= block_i$
6:    end for
7:    solve $ciphertext_i = K * str'_i$ for i=1 to d
8:    compute $K^{-1}$*ciphertext
9: until decryption make sense
```

number of iteration is $\frac{1}{ratio^{nd}} = 8^{nd}$

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

```
1: Part1:
Require: Ciphertext $Y_1, Y_2, ..., Y_n$
Ensure: K(mod2)
2: for all $\mu$ do
3:    compute $S_n(\mu) = \sum_y (-1)^{\mu.y} \times n_y$ where $n_y = \#\{k; Y_k = y\}$
4: end for
5: set all $\mu$ to the d values of $\mu$ with largest $S_n(\mu) = bias(\mu.Y)$
6: Part2:
7: for all $(i, i')$ do
8:    compute $n_{00}(i, i') = \#\{k < n : (\mu_i.Y_k, \mu'_i.Y_{k+1}) = (0, 0)\}$
9: end for
10: set $(i_d, i_1)$ to the first pair with lowest $n_{00}$
11: Part3:
12: for all $t = 2$ to $d - 1$ do
13:    for all i $\notin \{i_1, i_2, .., i_{t-1}, i_d\}$ do
14:       compute $n_{00}(i, i') = \#\{k : (\mu^T_{i_{t-1}} Y_k, \mu^T_i Y_k) = (0, 0)\}$
15:    end for
16:    take i such that $n_{00}$ is minimum and set $i_t = i$
17: end for
18: set $\mu = (\mu_{i1}, \mu_{i1}, ..., \mu_{id})$ and $K = (\mu^{-}1)^T$
19: output K
```

Here to be faster we store n_y in a table and we do a FFT on this table to get S_n. With this operation the total complexity drop from $O(d^2 \times 2^d)$ to $O(d \times 2^d)$ But it seems with some other techniques we could do better.

References

[1] S. Shazaei, S. Ahmadi. *Ciphertext- only attack on $d \times d$ Hill in $O(d13^d)$.*

[2] Alina, Matyukhina. *Cryptanalysis of the Hill Cipher.*

[3] Akavia, A. *Deterministic Sparse Fourier Approximation via Fooling Arithmectic Progressions.*

[4] Akavia, A., Goldwasser, S., Safra, S. *Proving Hard-Core Predicates Using List Decoding.*

[5] Hassanieh, H., Indyk, P., Katabi, D., Price, E. *Nearly optimal sparse Fourier transform.*

[6] Hassanieh, H., Indyk, P., Katabi, D., Price, E. *Simple and practical algorithm for sparse Fourier transform.*

[7] Iwen, M.A. *Combinatorial Sublinear-Time Fourier Algorithms.*

[8] http://www.nymphomath.ch/crypto/stat/anglais.html