IN, LASEC: Cryptanalysis of the Hill Cipher

Due on Spring 2016

Pr. Serge Vaudenay

Max Premi

June 10, 2016

Abstract

The author takes the previous paper about a new Ciphertext-only Attack on Hill Cipher, and try to improve it's complexity to get a better result than $O(d13^d)$.

The goal of this project is to improve the algorithm to get the key matrix modulo 26 and then to study the algorithm to recover the key matrix modulo 2, more precisely the Fourier Transform used.

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 algorithm and try to improve the complexity 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 look at the possible algorithm to improve the FFT in the recovery algorithm of matrix key modulo 2.

Hill 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], in time $O(26^{d^2})$.
Later, it has been proved that it is possible to get the key matrix K, with minimum length of ciphertext required $n = 8.96d^2 - O(\log d)$ [1] and the complexity has been reduced to $O(d26^d)$.
This method has been then enhanced [1] using the divide-and-conquer technique, by 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, we believe 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 $O(8^{nd})$, where n is the segment size.

Let's briefly recall 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 using the correlation between vectors is used to identify the last one and first one easily, and then recursively sort all the vectors.
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. However, we first need to deal with the improvement for translation of key matrix from $\mathbb{Z}/2\mathbb{Z}$ to $\mathbb{Z}/26\mathbb{Z}$.

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 getting the key matrix in $\mathbb{Z}/26\mathbb{Z}$ was possible, without considering $\mathbb{Z}/13\mathbb{Z}$.
First of all, we create a hash table using a very long text, and search mapping between segments of reference text and plaintext 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 segments 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:

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

It 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(# good matching) = (#segments in reference) $\times$ (#segment in plaintext) $\times$ $2^{-H_2(X)}$, 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 we are in $\mathbb{Z}/2\mathbb{Z}$:E(# all matching) = (#segments in reference) $\times$ (#segment in plaintext) $\times\, 2^{-H_2(X \bmod 2)}$. And indeed it is understandable that if 2 words modulo 2 are equal, these words are not always equal modulo 26.

As we consider independent letters in segment, the probability to find two of the same segment, with following occurrence of letters:$p(A) = 0.0808$, $p(B) = 0.0167$,...etc, is:$\Pr(a = b) = ((0.0808)^2 + (0.0167)^2 + ...)^n = 0.06609^n$. Then $H_2(X) = -\log_2 0.06609^n$. The frequencies given in [5] are used as reference.

Now let's consider E(# all matching). As only 2 values are possible it's easier than the previous, indeed E(# all matching) = (#segments in reference) $\times$ (#segment in plaintext) $\times\, 2^{-H_2(X \bmod 2)}$.
$H_2(X \bmod 2) = -\log_2(\sum_{i=0}^{1} \Pr(X \bmod 2 = i)^2)$, 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 (#segments in reference) $\times$ (#segment in plaintext) $\times$ 0.5^n.

Then to have an idea of the distribution of good matching and bad matching, the ratio $\frac{E(\#\text{good matchings})}{E(\#\text{all matchings})}$ is computed: $\frac{E(\#\text{good matchings})}{E(\#\text{all matchings})} = \frac{|\text{segments in reference}|\times|\text{segment in plaintext}|\times 2^{-H_2(X)}}{|\text{segments in reference}|\times|\text{segment in plaintext}|\times 2^{-H_2(X \bmod 2)}} = \frac{0.06609^n}{0.5^n} = \frac{1}{7.56^n} \cong \frac{1}{8^n}$

To decrease the actual complexity, we need to increase the ratio of good matching as E(# all matchings) can't be changed. So the only solution left is to try different assumptions and calculations for E(#good matchings).

An interesting case is to consider blocks of letters independents 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

What one can do, is taking the conditional entropy on the result of X mod 2.

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 $\alpha = 2$, $H_1(X \mid X \bmod 2 = y) = -\log_2(\sum_{i=1}^{n} \Pr(X = i \mid Xmod2 = y)^2)$
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:

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

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

and of course the same goes 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 [5] 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 $\frac{E(\#\text{good matchings})}{E(\#\text{all matchings})}$ in class of reduction modulo 2, so the conditional Expectation is as followed: $\frac{E(\#\text{good matchings}|X \bmod 2=y)}{E(\#\text{all matchings} \bmod 2=y)}$ or more clearly $\frac{E(\#\text{good matchings}|X\text{in class y})}{E(\#\text{all matchings in class y})}$. But in one class all the blocks are matching modulo 2 so it just gives Pr(#good matchings | X in class y) = $\Pr(X = i \mid X \bmod 2 = y)$
These result do not improve the ratio for low block size, but the interesting fact is that for blocksize $d > 10$, some blocks are alone in their classes reduction modulo 2, and we will use this property in the next part.

3.2 Computation with matching by class of reduction modulo 2

Let's now try to resolve the number of equations needed to 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 n good matching and the $d - n$ others are not.
We need $d - n < n$ meaning $n > \frac{d}{2}$ for the size of the segment.
If we call N the number of matching, and N_{good} the number of good matching, we get $2n - d$ equations in d^2 unknowns in fuction of N. What is the require number N of matching to be sure that we can compute the matrix?

Case $\frac{d}{2} < n < d$:

If $\frac{d}{2} < n < d$ then we got the following:

$$\binom{N}{N_{good}} \frac{13^{d^2}}{13^{2n-dN_{good}}} < N^{N_{good}} \frac{13^{d^2}}{13^{2n-dN_{good}}} < 1$$

That gives, considering that $N = rN_{good}$: $N_{good} > \frac{d^2 \log(13)}{(2n-d)\log 13 - \log r - \log N_{good}}$
The number of good matching needed is $N_{good} > 2d$ which is not exploitable with the current implementation.

Case $n \geq d$:

Now if $n \geq d$ then:

$$\binom{N}{N_{good}} \frac{13^{d^2}}{13^{dN_{good}}} < N^{N_{good}} \frac{13^{d^2}}{13^{dN_{good}}} < 1$$

Again as $N = rN_{good}$ we get: $N_{good} > \frac{d}{1 - \frac{\log r - \log N_{good}}{d \log 13}}$

From this we can say that N_{good} must be bigger than d, so $N_{good} > d$.

3.2.1 Method used to recover K_{26}

Thanks to the conditional entropy, we now know that some classes have a really good ratio of good matching/all matching. Indeed in a class of reduction modulo 2, a higher probability means that a lot of blocks are equal modulo 26.
Let's call the block modulo 2 *segment*, and block modulo 26 *string.*
We need to create a class table, and classify blocks by classes of reduction modulo 2, and by decreasing probability of apparition in these classes. This will be a hash table where buckets are segment modulo 2 assigned to different string of the reference text. We use a very long reference text to do so.
Let's assume we take one of the given ciphers Y_i, we compute, with the help of K_2, the plaintext modulo 2 X_i^2 and look to which class it belongs. We get C_k the class to which X_i^2 belongs.
We'll try to find a class where the ratio is 1, because if it's the case, it means that X_i^2 can only be equal to the string in the class C_k. From this we get, in the class C_k, the string corresponding to the segment X_i^2, and we get a first equation : $K \times X_i^{26} = Y_i$, where X_{26} is the string found in C_k.

From the previous computations we know that we need $d+1$ equations to solve the key matrix, the perfect case would be to take the $d+1$ classes where the probability that blocks are equals is 1. Experiment 3 shows that if $d \geq 12$ we have $d+1$ classes with Probability 1, meaning that there is only 1 block that can correspond to the mapping modulo 2.
We can resolve the $d+1$ equations by Gaussian reduction. It will take $O(d^4)$ time to solve the matrix. Experiment 4 gives details about this.

Condition on the number of blocks: The previous paper had a function to determine the minimum number n of cipher needed, in function of the size d of the matrix, to solve the key matrix modulo 2:

$$n = \frac{8}{\epsilon(1-\epsilon)^2} \times (d + log_2 d - log_2 p)$$

Where epsilon is the bias, and $p = p_1 - p_2$ where p_i is the probability that a pair of letter with the same position difference i from a random text is $(0,0)$.
For $d = 12$ we need approximately 8050 ciphers to be sure that the algorithm of key recovery modulo 2 works.
Then, we take the probability found by experiment on reference text to look at the probability that blocks are equals modulo 2:$8050 \times 0.0004043642104 = 3.22$ is the number of block expected to be equal to another in a class, for $d = 12$.
And indeed if we have 8050 ciphers, there a 2^{12} possibilities and $\frac{8050}{2^{12}} = 1.953$. It means that if all blocks are equally distributed, there would be approximately 2 blocks by class of reduction modulo 2, but as it's not an equal distribution, we say that the average number of blocks in a class is:

$$n \times \Pr(\text{Collision modulo 2})$$

The probabilities Pr(Collision modulo 2) are computed thanks to a reference text and result are displayed in Experiment 2.

An algorithm is implemented in the Appendix to find the key matrix modulo 26 with a complexity of

$O(k^d \times d^{3.8})$ where k is the number of block estimated in the classes. We expect in our algorithm to have only 1 block in the best classes as Experiment 3 shows it. Else, we do it for the first string k that appears with the highest probability.
We need $k^d \times d^{3.8} < 13^d$ for our algorithm to be efficient, $k < \sqrt[d]{\frac{13^d}{d^{3.8}}}$.

So we have found an algorithm that works for blocksize $d > 11$ that runs in a polynomial time and is better than $O(13^d)$ if the plaintext is unique in his class, and if the cipher has less than $\sqrt[d]{\frac{13^d}{d^{3.8}}}$ different blocks in his class.

4 Study of Faster Fourier Transform for Algorithm 1

We want to enhance the possible FFT on a table called n_y which contains the number of times where each cipher Y appears. So it is a table containing numbers $\in \mathbb{N}$ of size $N = 2^d$.
In this section, we will study possible algorithm that can improve the complexity of the FFT, with the help of Sparse Fourier Transform.
When a fast Fourier Transform (FFT) is applied to a signal of size N, the complexity in time is $O(N \log N)$.
A general algorithm for computing a 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.

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 it takes exactly $O(klog^2 l)$, but in general to estimate x takes approximately $O(\sqrt{lk})$.
This algorithm perform better if:

$$\sqrt{2^d k}(\log 2^d)^{\frac{3}{2}} < 2^d \log 2^d$$

Meaning $k < \frac{2^d}{\log 2^d}$ 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 recent algorithm 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 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

This is also an SFT algorithm, meaning that it'll output the k-non null coefficient of the matrix. To do so you need to give it a threshold $\tau \in (0, 1]$ and an oracle access to a function f, it outputs the τ-significant Fourier Coefficient. It 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. A τ-significant Fourier Coefficient is a coefficient whose magnitude is at least a τ-fraction of the sum of squared Fourier Coefficient.
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, an we're testing for the first two if the norm of $\hat{f}$ the Fourier Transform squared is equals to the set_i oracle output squared. Meaning more explicitly : $\hat{f}(J_i)^2 = \sum_{\alpha \in J_i} |\hat{f}(\alpha)|^2$. 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.

We define the semi norm as:$||f||_2 = \sqrt{\sum_x |f(x)|^2}$ The threshold τ can be chosen, with the fact that a α is a $\tau - significant$ Fourier coefficient if and only if $|\hat{f}|^2 \geq \tau ||f||_2^2$
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 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 $\lfloor \frac{n}{2^d} \rfloor^2$ ciphers by buckets, and the worst case where the semi norm is the bigger is when all ciphers 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 $\lfloor \frac{6200}{2^{10}} \rfloor = 6$, we get $|\hat{f}|^2 \geq \tau||f||_2^2 = (2^d \times \lfloor \frac{n}{2^d} \rfloor)^2 \geq \tau 2^d \times \lfloor \frac{n}{2^d} \rfloor^2 = 6144^2 \geq \tau 36864$, where 6144 is the biggest Fourier coefficient and all the others are 0 so $1 = \tau$ that is a good result for evenly distributed cipher. It works for almost evenly distributed ciphers, as the DFT will have small pikes and will be considered as zero.

So this works for almost evenly distributed ciphers, and a complexity of $O(log(2^d))$ will be achieved.

Experiment

Experiment 1:Probability of independent English letters

From the frequency letter given by Wikipédia, in english we got the following result :
Probability summed = 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 $(\sum_{i=0}^{26} \Pr(i = 1)^2)^n, i \in \{alphabet \bmod 2\}$
Ratio of good matching and all matching=0.1285934407027314^n
From the Rényi entropy seen previously we get, $\frac{|\text{segments in reference}| \times |\text{segment in plaintext}| \times 2^{-H_2(X)}}{|\text{segments in reference}| \times |\text{segment in plaintext}| \times 2^{-H_2(X \bmod 2)}} = \frac{0.06549^n}{0.5^n} = \frac{1}{7.776^n}$ So $\frac{1}{7{,}77644^n}$ which match the result found.

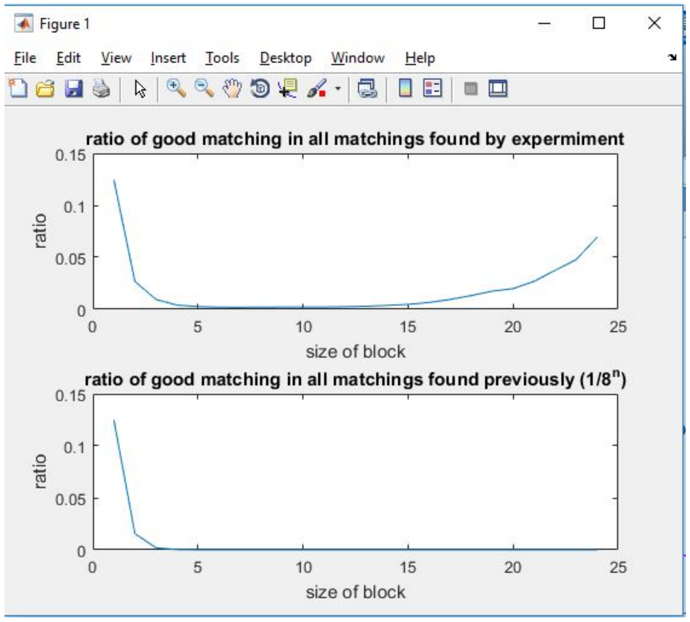
Another site [5], with a total number of 100000 letters composed with texts from Edgar Allan Poe, Arthur Conan Doyle, and 4 articles from encyclopedia Encarta 95:

Probability summed = 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 $(\sum_{i=0}^{25} \Pr(i = 1)^2)^n, i \in \{alphabet \bmod 2\}$
Ratio of good matching and all matching=0.12996168115565054^n
From the Rényi entropy seen previously we get, $\frac{|\text{segments in reference}| \times |\text{segment in plaintext}| \times 2^{-H_2(X)}}{|\text{segments in reference}| \times |\text{segment in plaintext}| \times 2^{-H_2(X \bmod 2)}} = \frac{0.06609^n}{0.5^n} =$

$\frac{1}{7.69n}$ So $\frac{1}{7{,}69457^n}$ which also match what we expected.

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 base all computation on the number of time it appears. So the basic is to choose a blocksize, then it'll count every different blocks that appears modulo 26 and also convert them in modulo 2. Then it'll compute the probability that a good matching appears 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 complexity depends on the ratio to ther power blocksize, so it will not be good enough to be taken into account.

We also have the probability of collision modulo 2 that are used to estimate the number of blocks in a class modulo 2.

blocksize	Probability of collision modulo 2
1	0.50828
2	0.264899
6	0.0196614
10	0.0014748
11	0.00072321
12	0.000404364
13	0.0002116537
14	0.0001106218
15	0.00005764109
16	0.00002986585

This represents the probability that a block is equal to another modulo 2.

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

We now count the number of classes where the probability that blocks are equal = 1, or more formally, classes where there are only 1 block in it.

blocksize	Number of classes where Probability of good matching =1
1	0
10	3
11	8
12	12
13	18
14	29
15	48
16	78

Experiment 4:Example of the attack

So let's assume we take $d = 3$ (to simplify the calculations) and we find $d + 1 = 4$ pairs ciphers/plaintext modulo 26 with probability 1 for each.
The pairs plaintext,ciphertext are :

$\begin{pmatrix}1\\2\\7\end{pmatrix}, \begin{pmatrix}15\\24\\24\end{pmatrix}$

$\begin{pmatrix}2\\4\\8\end{pmatrix}, \begin{pmatrix}24\\20\\6\end{pmatrix}$

$\begin{pmatrix}3\\5\\1\end{pmatrix}, \begin{pmatrix}22\\18\\10\end{pmatrix}$

$\begin{pmatrix}2\\1\\3\end{pmatrix}, \begin{pmatrix}10\\15\\23\end{pmatrix}$

If we call the key matrix modulo 26, K_{26} it'll look like this:

$\begin{pmatrix}x_1 & x_2 & x_3\\x_4 & x_5 & x_6\\x_7 & x_8 & x_9\end{pmatrix}$

From this we get d system with $(d + 1)$ equations which are the following for the first row:

$$\begin{cases} x_1 + 2x_2 + 7x_3 \bmod 26 = 15 \\ 2x_1 + 4x_2 + 8x_3 \bmod 26 = 24 \\ 3x_1 + 5x_2 + x_3 \bmod 26 = 22 \\ 2x_1 + x_2 + 3x_3 \bmod 26 = 10 \end{cases}$$

We can solve these systems by Gaussian reduction, it'll take d^3 for each system, so the total complexity will be $O(d \times d^3)$ which is lower than $O(13^d)$. The problem is to find each time one cipher that match the block modulo 2.
As we apply an invertible matrix, the distribution of segment ciphers should be identically independent, as the segment of the plaintext are considered identically independent. Therefore, we can take the probability that 2 segments are equals modulo 2 from the plaintext, it should be around the same for the cipher segments. It brings the complexity of key matrix's recovery modulo 2 to $O(d)$ only if it's evenly distributed.

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 from the 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 $n_{00}(i, i') = \#\{k : (\mu_{i_{t-1}}^T Y_k, \mu_i^T Y_k) = (0, 0)\}$
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.

Algorithm to solve the matrix key modulo 26.

Require: Ciphertext $Y_1, Y_2, ..., Y_n$ and classification of ciphers and plaintext in classes C_i of reduction modulo 2

```
1: Find the $d+1$ classes with the highest probability that a block appears for the plaintext.
2: for all $d+1$ classes $C_i$ take the $block_i$ associated modulo 26, $i \in \{0,1,...,d+1\}$ do
3:     find the cipher modulo 2 called $Y_i^2$ associated to the $block_i$
4: end for
5: repeat
6:     take one of the string $k \in C_i$, called $X_{i_k}26$ that match with $Y_i^2$,$i \in \{0,1,...,d+1\}$
7:     resolve $K \times X_{i_k}26 = block_i$, for all i.
8: until it makes sense
```

complexity is $O(|ref|+|plain|+2^d+k^d \times d^{3.807})$ We use Strassen Algorithm to multiply the matrix, it brings the complexity from d^3 to $d^{2.807}$

Conclusion

In this paper we presented a possible improvement for an ciphertext-only attach on Hill cipher using the distribution of ciphers and plaintexts in $\mathbb{Z}/26\mathbb{Z}$ in function of their reduction in $\mathbb{Z}/2\mathbb{Z}$. The complexity of key recovering in $\mathbb{Z}/2\mathbb{Z}$ is $O(d)$ only in the restrictive case where the ciphertexts are evenly distributed in class of reduction modulo 2. Else it keeps it's higher bound $O(d \times 2^d)$
In $\mathbb{Z}/26\mathbb{Z}$, we found a better solution for blocksize $d > 11$, by using the classification of cipher and plaintext in classes of reduction modulo 2. After comparing with experimental result it gives us more blocks with Probability of collision $= 0$, on higher blocksize.
The complexity found is $O(k^d \times d^{3.807})$ and is better than $O(13^d)$.
Future work- reducing the complexity of the FFT in a better way by finding how to be sure that a vector will give a Sparse FFT.

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

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