Cracking the playfair cipher

Folmer Heikamp

June 5, 2020

Abstract

The playfair cipher is a matrix based cipher which was invented in 1800 and named after Lord Playfair. It is a substitution cipher based on bigrams. In this paper several methods for cracking the playfair cipher are given. Inspiration for writing this report came from exercise 6 from the code book, written by Simon Singh.

1 Introduction

In the code book **??**, written by Simon Singh, there are several exercises. Each exercise challenges you to crack a ciphertext. Exercise 6 is described as follows:

A quick examination shows that it is encrypted with the Playfair cipher (no 'j'). Cracking it is quite some work and I wondered if it could be done automatically. Wikipedia [] said that it could be cracked using shotgun hill climbing and some projects on github[**?**, **?**] were able to crack the Playfair cipher by using simulated annaeling and genetic algorithms respectively. I wondered if I could reproduce the results.

The playfair cipher was invented by Charles Wheatstone in 1854. The cipher gets its name from the foremost promotor: Lord Playfair. The cipher is a bigram mono-alphabetic substitution cipher which means that every plaintext bigram is mapped to a ciphertext bigram.

The cipher is not considered safe by today's standards but cracking it is not straightforward. A monogram mono-alphabetic substitution cipher is easily cracked using frequency analysis. However when bigrams are used frequency analysis becomes harder and more error prone since there is now a mapping from $A^2 - > A^2$ where A is the alphabet containing 25 characters ('j' is missing). It can be cracked manually but it takes some effort.

The main goal of this article is to show that the cipher can be cracked automatically. To do this several methods are explored. The methods used are shotgun hill climbing, simulated annaeling and genetic algorithms. All three are meta-heuristic aimed to find an optimal solution in a large searchspace. This article also describes a algorithm which helps in finding a solution manually. The second goal is to do a bit of experimentation with parameter settings and plot the results.

Figure 1: Matrix initialized with keyword 'gorilla' (notice 1 l)

The layout of this article is as follows. Section 2 contains background information about the Playfair cipher and metaheuristics. Section 3 describes the cracking methods(including the helper algorithm) and some examples. Section 4 contains the experiment setup. The results from the experiment are given and analyzed in section 5. This report will finish with a conclusion and future work.

2 Background

I will try to explain all the concepts used in this article on a basic level. However I am not an expert on all of these topics nor is it necessary to explain concepts better explained by other people. When needed I will refer to other books and articles.

2.1 Playfair

The playfair cipher uses a matrix to do encryption and decryption. This matrix has five rows and columns and is initialized using a keyword. The matrix contains all possible characters which can be used in the text. Because of this only 25 character can be used. Since we have 26 characters in the latin alphabet we have to remove one character. Usually this is solved by replacing 'j' with 'i'. The matrix is initialized by removing duplicates in the keyword and appending all characters in the alphabet not in the keyword. An example with keyword 'gorilla' can be seen in figure 2.1. Playfair has a rule that bigrams with the same symbol (aa, bb, etc.) should not be encrypted together. To solve this an 'x' is placed between them. If the message lenght is odd an 'x' is appended to the text to take care of the final bigram. Notice the difference between 'aacd' and 'caad'. The first one becomes 'ax ac dx' (no duplicates and odd length), while the last becomes 'ca ad' (aa is not a bigram).

The encryption of a bigram is based on three rules. Look up the characters in the keysquare.

1. If they are in the same row, replace with the character to the right. Wrap around if needed.

2. If they are in the same column, replace with the character below. Wrap around if needed.

3. If the above does not apply: form a square and use the other 2 characters. Character in the same row as the first character in the plaintext is the first in the ciphertext.

Decryption is basically the same, only reversed.

As can be seen the bigram frequencies stay about the same ('j' and 'x' make it somewhat different), which means that just as with monogram mono-alphabetic ciphers, frequency analysis can be used. This is a big indicator that the cipher is not safe to use.

For more information about the playfair cipher [**?**].

2.2 Meta-heuristics

We will not go in too much detail about meta-heuristics, since there are a lot of them. The basic idea behind all of them is to find the best solution in a searchspace where it is unfeasible to search it all. Meta-heuristics only work in problems where you can define a fitness function which can tell you if a solution s_1 is better or worse than a solution s_2. For this reason a meta-heuristic should work on Playfair but not on AES. To see this consider two keys k_1 and k_2, where k_2 is the same as k_1 only one character is swapped. In Playfair the resulting plaintext for both keys would be about the same. The one with which matches the best with the known frequencies is probably best. If you apply the keys to AES you will get two entirely different messages and you have no way to determine which one is best. Finding a working fitness function for AES means that is is broken.

A problem with meta-heuristics are local optima. Therefore the algorithm must have some means for breaking out of a local optimum.

We use three different meta-heuristics. Two of them are local search methods and the other is a genetic algorithm.

2.2.1 Shotgun Hill Climbing

Shotgun hill climbing is a non-stochastic local search method. It is basically hill climbing with random restarts. The shotgun refers to the fact that you simply try a lot of possible random starts. Shotgun hill climbing is easy to paralelize. The basic approach of hill climbing is: choose a random starting node. Calculate all the possible neighbouring nodes. Select a random neighbour node and set it as the current node if the score is greater than select the node, otherwise continue. This is done for N iterations or if the last K iteartions no solution was found. Shotgun hill climbing tries to overcome local optima by using random restarts. The other methods used in this report use more sophisticated methods for overcoming local optima.

2.2.2 Simulated Annaeling

Simulated annaeling is a bit like hill climbing. The main difference is that SA is stochastic. Less performing solutions are sometimes accepted. This prevents the algorithm getting stuck in local optima. This is in contrast to shotgun hill climbing which uses random restarts for this.

2.2.3 Genetic Algorithms

Genetic algorithms are inspired by evolution by natural selection. It is an iterative process and just like local search methods it tries to find the best solution for a given problem. In GA a population is generated randomly. From this population a new population is constructed. This is done by using selection, crossover and mutation. First all the individuals in the population are scored. After that a subset is selected from the population where better scoring individuals have more chance of being selected. From this selection a new population is constructed using crossover. Crossover works by selecting two parents and generate offspring from them. The child is a combination of both parents. Crossover is done until the new population has the same size as before. After this the new population is mutated. Mutations occur infrequently but enough to prevent the algorithm getting stuck in local optima. There is a lot variation within GA's. For a more detailed explanation read [].

2.3 Cryptanalys

The methods in this article are all based on frequency analysis. In some ciphers the frequencies stay the same. This leads to a possible attack. For example take a caesar cipher with shift 3. This means that 'e' becomes 'h' in the ciphertext. 'e' is the most common language so in the ciphertext 'h' is probably the most occurring character. From this the crypto-analyst can assume that 'e=h'. Cryptanalysis is too broad a topic to be discussing here in detail. For a more detailed description of cryptanalysis check out [].

3 Cracking

To crack the playfair cipher three solutions were proposed. The first approach is called 'helper' and is developed to help the attacker keep track of bigram mappings. It also suggests the most likely bigram mappings. It is not an automated attack.

The second approach consists of two algorithms. Both are local search algorithms.

3.1 Helper

A monoalphabetic substitution cipher is usually solved using frequency analysis. You can do the same for bigrams but than it is much harder because there are about 600 different bigrams []. Cracking it manually requires a lot of paperwork. To facilitate this a algorithm was developed. The basic idea is to sort all the ciphertext bigrams on frequency, descending. The sorted list is compared with known plaintext frequencies in English. For a given ciphertext bigram c and a table with known frequencies P the basic score between c and $p, p \in P$ is given by $|frequency(c, text) - p|$.

There are several observations which can be used to make a better choice.

Reversed bigrams If $x = x_1x_2$ is mapped to $y = y_1y_2$ then x_2x_1 must map to y_2y_1. This property can be used in the score. For example if x maps to 're' but the reverse of x does not occur often in the ciphertext than it the mapping is probably incorrect since 'er' should also occur reguraly.

In the square If

3.2 Local Search methods

We used two local search methods, shotgun hill climbing and simmulated annaeling. The shotgun hill climbing approach is very simple and is given as pseudo code in **??**.

```
INPUT: max\_iterations , stopcriterium , ciphertext
OUTPUT:
k <- randomly generated key
score <- score k by decrypting ciphertext with k and summin
```

The problem with this method however is that it easily gets stuck in a local optimum. To solve this the number of iterations has to be set to a big number. For our purpose we set the number of iterations to 10000000 and we stop after 200000 steps without solution or if the maximum number of iterations is reached. Some experimetation shows that it is not easy to bypass the local optima. Another approach is to do some more random restarts with less iterations.

The other approach uses simulated annaeling. We tried the algorithm with the simple scoring algorithm using trigrams. A run of the program shows that it goes into a local optimum very soon. However because of it stoachastic nature it occasionally breaks out and improves by quite a bit. With some moderate mutation settings and a bit of time it is able to decrypt a test message. This concludes the proof of concept for simulated annaeling.

3.3 Genetic algorithm approach

We need two things, a genetic representation for playfair and a proper fitness function. The genetic representation is simple: a string of 25 characters (a-z, except j) each character only occuring once. The fitness function is the same as for the local search methods, see **??**. In our example we will use trigram frequency to crack a message. It also uses the same mutations as in the local search methods, only the probability of mutation is much lower because it serves a different purpose. In the other methods mutation is the only way for generating new keys. In genetic algorithms it is subsidiary to cross-over. The reason why there is mutation is to prevent that the algorithm stays in a local optimum. The cross-over operator is a bit difficult. Normally you would just do 1-point crossover or 2-point crossover. However that would lead to duplicates in the offspring. To prevent this the cross-over is implemented as such:

1. Select 2 parents.
2. Select a random index i.

3. Swap the character at index i in the parents
4. If it leads to a duplicate: find the other index j in one of the parents and go to three
5. If no duplicates return the 2 generated offspring

Example Consider the following ciphertext:

4 Experiments

We only experimented on the genetic algorithm. There are several questions which could be answered. First of all which mutations deliver the best results. Secondly which fitness function delivers the best results.

4.1 Experiment 1

Experiment focuses on mutation rates. The settings for each are as follows:

Restarts 10

| content... |

4.2 Experiment 2

Experiment 2 focus on two scoring functions. The simple scoring function and the GTest.

5 Future Work

- More fitness functions. The IOC is an indicator of a language. IOC can also be generalized to include bigrams. A possible fitness function is $|IOC(text) - IOC(english)|$, where $text$ is the ciphertext decrypted with a possible key k. You could add bigrams as well. Another possibility is to use ranking instead of actual frequencies. Another possibility is to use another metrics than frequencies. Is is unclear what kind of measure you could use.
- Multiprocessing. Random restarts could be done in parallel, however my implementation uses ctypes with pointers which the python multiprocessing library does not like because the class is not pickable. There should be a work around.
- Possibly this approach also works for the adfvgx and the adfgx ciphers.

6 Recommendations

There is a fun way to make this cipher stronger in a vigenere like way. Pick a key sentence and construct for every word longer than $n(n > 8$ should be adequate) a key in the playfair way. Encryption works by iterating through the playfair keys just as in vigenere ciphers. An attack should work along the way of the vigenere cipher. Calculate kasiski distance or IOC and solve for every range.

7 Conclusion

All attacks presented in this paper work reasonably which means that the solution is usually found. All methods are heuristics so an solution might not be found. The attacks only work if the language in use is English. The genetic algorithm especially is often in a local maximum, this can be solved by random restarts.

A lot of improvements can be done but it has to be considered if it is worth the time. The Playfair cipher is not used in practice anymore so the probability that the knowledge of how to crack a Playfair cipher is useful is neglible.

The code can be found on github [].

Make it poly-alphabetic Extension to other square ciphers and ciphers in general. Fitness for AES is hard. Dependence on frequency Fout in de software. Om een of andere manier voegt de decrypt soms een extra symbool toe.

A Shotgun Hill Climbing

Ran the algorithm twice on the stage_6 ciphertext, see appendix **??** and the new restart with key geidkbpztoxnrwuyflhvscqam key geidkbpztoxnrwuyflhvscqam with score: 0.000071 best score: 0.000074 for key geidabpztoxnrwuyflhvscqkm (iter: 1) best score: 0.000090 for key bpztogeidaxnrwuyflmvscqkh (iter: 3) best score: 0.000092 for key bpotzgeadixnuwryfvmlschkq (iter: 9) best score: 0.000103 for key bpotzgerdixnuwayfvmlschkq (iter: 17) best score: 0.000109 for key bpitzgerdoxnuwayfvmlschkq (iter: 43) best score: 0.000117 for key xpitzgerdobnuwayfvmlschkq (iter: 47) best score: 0.000118 for key xpitzgerdobnuwayfcmlsvhkq (iter: 58) best score: 0.000119 for key xpiszgerdobnuwayfcmltvhkq (iter: 79) best score: 0.000132 for key ypiszterdoxnuwagfcmblvhkq (iter: 99) best score: 0.000143 for key ypiszterdmxnuwagfcoblvhkq (iter: 100) best score: 0.000152 for key ypirztesdmxnuwagfcoblvhkq (iter: 118) best score: 0.000175 for key ypirztesdmwnuxagfcoblvhkq (iter: 136) best score: 0.000190 for key ypirztesdhwnuxagfcoblvmkq (iter: 161) best score: 0.000194 for key ypirktesdhwnuxagfcoblvmzq (iter: 168) best score: 0.000194 for key ypirctesdhwnuxagfkoblvmzq (iter: 198) best score: 0.000199 for key ypirctesdhwnuxagfkomlvbzq (iter: 211) best score: 0.000201 for key ypirctesdhwnuoagfkxmlvbzq (iter: 250) best score: 0.000202 for key ypirctesdhwnuoagfkxqlvbzm (iter: 267) best score: 0.000213 for key yporctesdhwnuiagfkxqlvbzm (iter: 291) best score: 0.000215 for key yporctesdhwnuiagfkmqlvbzx (iter: 336) best score: 0.000217 for key yporctesdhwnuiagfkmqlvbxz (iter: 449) best score: 0.000221 for key yporktesdhwnuiagfcmqlvbxz (iter: 472) best score: 0.000234 for key lporktesdhwnuiagfcmqyvbxz (iter: 510) best score: 0.000237 for key gfcmqtesdhwnuialporkyvbxz (iter: 532) best score: 0.000237 for key gfcmqtesdhwnuialporkyvzxb (iter: 692) best score: 0.000244 for key hfcmqtesdgwnuialporkyvzxb (iter: 1392) best score: 0.000247 for key hfcmgtesdqwnuialporkyvzxb (iter: 1401) best score: 0.000250 for key hfcmgtesdqwnuialporkzvyxb (iter: 1466) best score: 0.000258 for key hfcmgtesdywnuialporkzvqxb (iter: 1485) best score: 0.000275 for key hfcmgtesdyunwialporkzvqxb (iter: 1558) best score: 0.000287 for key hcfmgtsedyuwnialoprkzqvxb (iter: 1600) best score: 0.000294 for key hcfmntsedyuwgialoprkzqvxb (iter: 1601) best score: 0.000294 for key hcfmntsedyuwgialoprkbqvxz (iter: 1616) best score: 0.000300 for key hcfmntsedyuwgiolaprkbqvxz (iter: 1729) best score: 0.000300 for key hgfmntsedyuwcioaprklbqvxz (iter: 1747) best score: 0.000393 for key hgfzotsedluwcimaprknbqvxy (iter: 1756) best score: 0.000396 for key hgfmotsedluwcizaprknbqvxy (iter: 1765) best score: 0.000398 for key hgfmbtsedluwcizaprknoqvxy (iter: 1811) best score: 0.000402 for key hyfmbtsedluwcizaprknoqvxg (iter: 1838) best score: 0.000404 for key hyfzbtsedluwcimaprknoqvxg (iter: 1844) best score: 0.000421 for key hvfzbtsedluwcimaprknoqyxg (iter: 1888) best score: 0.000432 for key hvfzbtsecluwdimaprknoqyxg (iter: 1980) best score: 0.000435 for key hvfzbtsecluwdimoqyxgaprkn (iter: 2030) best score: 0.000435 for key hqfzbtsecluwdimovyxgaprkn (iter: 2037) best score: 0.000451 for key hqfzbtsecluwdimyvoxgaprkn (iter: 2041) best score: 0.000452 for key hqbzftslceuwmidyvgxoapnkr (iter: 2422) best score: 0.000460 for key hqbzftslceuimwdyvgxoapnkr (iter: 2428) best score: 0.000480 for key hqbzftslceuimwdyvgxoaknpr (iter: 2484) best score: 0.000483 for key hqbzftslceuimodyvgxwaknpr (iter: 2636) best score: 0.000485 for key hvbzftslceuimodyqgxwaknpr (iter: 3010) best score: 0.000487 for key hvbxftslceuimodyqgzwaknpr (iter: 3066) best score: 0.000590 for key hvbxfpqgzwuimodtslceaknyr (iter: 43328) best score: 0.000594 for key hvbxfpqgzduimowtslceaknyr (iter: 43348) best score: 0.000595 for key hvbcfpqgyduimxwtslzeaknor (iter: 43356) best score: 0.000598 for key hvbcfpqgyduimzwtslxeaknor (iter: 43377) best score: 0.000603 for key hvbcfpngyduimzwtslxeakqor (iter: 43423) best score: 0.000617 for key hvbxfpngoduimcwtslyeakqzr (iter: 43428) best score: 0.000617 for key hvzxfpngoduimcwtslyeakqbr (iter: 43732) best score: 0.000628 for key hvzbfpngxyuimowtslceakqdr (iter: 43836) best score: 0.000699 for key hvzbfpngxycimowtslueakqdr (iter: 43854) best score: 0.000723 for key hvzbfpngoycimxwtslueakqdr (iter: 43872) best score: 0.000734 for key hvzbfpngoycqmxwtslueakidr (iter: 43940) best score: 0.000777 for key hvzbfpngiycqmxwtslueakodr (iter: 43969) best score: 0.000785 for key hvzbfpngiycqxmwtslueakodr (iter: 44050) best score: 0.000825 for key hnzfbpvgyicqxwmtsleuakord (iter: 44056) best score: 0.000845 for key hnzfkpvgyicqxwmtsleuabord (iter: 44078) best score: 0.000849 for key hnzfkpvgyimqxwctsleuabord (iter: 44102) best score: 0.000851 for key hncfkpvgyimqxwztsleuabord (iter: 44107) best score: 0.000857 for key hncfkpvgyizqxwmtsleuabord (iter: 44182) best score: 0.000919 for key hncfkpvgyiqxwmztsleuabord (iter: 44218)

```
best score: 0.000926 for key hncfkpvgxiqywmztsleuabord (iter: 44219) best
score: 0.000929 for key hncfkpzgxiqywmvtsleuabord (iter: 44250) best score:
0.000934 for key hnwfkpzgxiqycmvtsleuabord (iter: 44307) best score: 0.000947
for key hnwfkpzgxiqvcmytsleuabord (iter: 44380) best score: 0.000959 for key
hnbfkpzgxiqvcmytsleuaword (iter: 44423) best score: 0.000967 for key hnbfkpzg-
miqvcxytsleuaword (iter: 44430) best score: 0.000982 for key hnbfkpzgmiqcvxyt-
sleuaword (iter: 44552) best score: 0.000991 for key hncfkpzgmiqbvxytsleua-
word (iter: 44556) best score: 0.000993 for key hncfkpygmiqbvxztsleuaword
(iter: 44579) best score: 0.000996 for key hncfkpygmibqvxztsleuaword (iter:
44680) best score: 0.001011 for key hycfkpngmibqvxztsleuaword (iter: 44697)
best score: 0.001147 for key hycfkpigmnbqvxztsleuaword (iter: 44830) best
score: 0.001164 for key hbcfkpigmnyqvxztsleuaword (iter: 44920) best score:
0.001169 for key hbcfypigmnkqvxztsleuaword (iter: 44968) best score: 0.001188
for key hbcfypigmnzqvxktsleuaword (iter: 45003) best score: 0.001235 for key
hbcfgpiymnzqvxktsleuaword (iter: 45027) best score: 0.001546 for key hbcfg-
pikmnzqvxytsleuaword (iter: 45137)
```