DAT 510 Assignment 1)

Abstract

The focus of this project was to solve problems related to implementing and cracking cryptographic ciphers. The project consisted of two main parts, the first of which was related to polyalphabetic ciphers. In this part the main task consisted of developing tools to analyze and crack encrypted text for which the specific method and key from which the encrypted text was produced, was unknown. This task was executed successfully, where the cipher method was identified as an *autokey-cipher,*with the key being found as: “DATFBA”. The second part consisted of implementing and utilizing simplified DES, a lightweight model-version of the larger DES (data encryption standard), on which we were to encode and decode binary strings. Followingly, TripleSDES a threefold composition of DES ciphers, was also implemented. From these implementations a part of the task was to find the key on which two binary messages was encoded on each respective algorithm. Using a brute force approach, the key was found to be “11111010” for the SDES encoded message and “11111010--0101011111” for the TripleSDES encoded message.

Introduction

The field of cryptography is vital for the modern data and communication technology, being the primary source of means for ensuring secure communication, privacy, and data security. To get some semblance over how methods from this field is utilized in the real word, this project consists of solving simplified but relevant problems. More specifically, these problems consist of implementing and cracking different cipher algorithms.

Design and Implementation

Part 1 Cracking a polyalphabetic cipher

Task 1.

Of all the tasks in the project, this task proved to be the most challenging and time consuming. At first an attempt was made to decode the text as a Vigenère cipher, but this yielded no intelligible results. After this attempt, an implementation for the autokey cipher algorithm yielded the correct result. The general method for performing cryptanalysis on such a cipher is described on Wikipedia [1]. Hence the task was to create the necessary tools to be able to perform this analysis. The solution can be summarized by first describing the general method, and then describe the specific implementation.

An autokey substitution cipher is a polyalphabetic substitution-cipher which is like the Vigenère cipher in that it consists of a plaintext and a key of equal length, where the characters in both texts belong to a given alphabet. The characters in the alphabet can then be treated as numeric values, and the result of the cipher is the sum of the key and plaintext value. Where autokey differs from Vigenère is when how the key may be generated. In the case of Vigenère the key may be a simple keyword repeated throughout the length of the plaintext. With this type of encoding, statistical characteristics of the language is preserved for every key-length spaced letter, and cryptanalysis can be performed by bucketing the text and analyzing the statistical similarity to English text. The autokey cipher works by appending a keyword to the front of the plaintext and using this as the key. The advantage of this method is that the statistical properties of the text is not preserved, and the statistical methods one would use in the Vigenère case will not work. Rather, the strategy for cracking an autokey cipher can be inducted by first realizing that the ciphertext itself is most of the key in the cipher. If one were to guess or know a word of length n which appears in the plaintext, one would also know that this word encodes the next n letters a keyword-length further down the plaintext. Furthermore, one would also be able to find the word which together with the ciphertext n letters up, would produce the guessed/known word. From this fact, one can lead the following strategy: Choose a word/n-gram that is likely to appear in the plaintext such as “the, and, tha, …” then for key-lengths up to a set limit, decode the next and previous n-gram one key length apart. If the n-grams that results from the decoding seems likely to appear in the plaintext one can repeat the process. Should one guess correctly, then it is likely that enough of the plaintext is revealed to be able to surmise the contents.

The main component of the solution is the python class Autokey\_cryptanalysis, which is a class which stores the given ciphertext and methods to process and develop this text into the sought solution.

CRYPTOGRAPHY CAN BE STRONG OR WEAK CRYPTOGRAPHIC STRENGTH IS MEASURED IN THE TIME AND RESOURCES IT WOULD REQUIRE TO RECOVER THE PLAINTEXT THE RESULT OF STRONG CRYPTOGRAPHY IS CIPHERTEXT THAT IS VERY DIFFICULT TO DECIPHER WITHOUT POSSESSION OF THE APPROPRIATE DECODING TOOL HOW DIFFICULT GIVEN ALL OF TODAYS COMPUTING POWER AND AVAILABLE TIME EVEN A BILLION COMPUTERS DOINGA BILLION CHECKS A SECOND IT IS NOT POSSIBLE TO DECIPHER THE RESULT OF STRONGCRYPTOGRAPHY BEFORE THE ENDOF THE UNIVERSE

With the keyword “DATFBA”

Part 2. Simplified DES

Task 1 and Task 2

The main task in this part was to code an implementation of the simplified DES (SDES) algorithm in concordance with the given description [2]. This algorithm is as the name suggest a simplified version of the DES (Data Encryption Standard) block cipher algorithm. This implementation was further used to create the triple des algorithm which in essence takes two keys, and essentially chains together three encryption rounds of the SDES algorithm. With these implementations a table of keys, encrypted and decrypted binary string were filled in. The results of this are as follows:

Task 3

Task 3 gives two strings of binary digits, which are the results of converting ascii characters to binary and encoding both strings with their respective of the two algorithms, where the keys are unknown. The task was then to attempt to crack the ciphers and retrieve the decrypted plaintext and the encryption key. The strategy used to solve this task was a simple brute force attack. In the SDES algorithm, the key can be a value between 0 and 1024 in binary. One can then check for every value in the range if it is a potential key. One can check this by choosing a word that is likely to appear in the plaintext, in this case the word “des” was chosen. Since the output of the SDES algorithm vary strongly dependent on the key and plaintext, and since the output will be encoded into ascii which is a relatively larger set of characters, it is unlikely for the search word to appear in an incorrect decryption from a false key. By using this strategy, and by incidentally choosing a correct keyword, the plaintext was found to be:

“Simplified des is not secure enough to provide you sufficient security”

With key: [1, 1, 1, 1, 1, 0, 1, 0, 1, 0]

The case for TripleDES is similar, however in this case the range for potential key is much greater, in fact it is the range of SDES squared which accounts for over a million key to check. Account the fact that it takes some time to perform the encryption chain of the algorithm, brute forcing the triple des algorithm takes a significantly greater amount of time. However, with patience and a relatively powerful computer, the same strategy works without augmentations to the method. The message was found to be the same as previous with the keys:

[[1,1,1,1,1,0,1,0,1,0], [0,1,0,1,0,1,1,1,1,1]]

Task 4:

In this task a small we server was set up in in which takes binary strings as input into the TripeDes algorithm and yields the decrypted text. This reflects a communication protocol which utilizes TripleSDES as the cipher, which has been demonstrated in this project to be not secure as it is susceptible to brute force attacks.

[1] <https://en.wikipedia.org/wiki/Autokey_cipher>

[2] William Stallings Cryptography and Network Security, Fifth Edition Prentice Hall 2010 ISBN-10:0136097049

[3] 2013 - 2019 Daniel Rodriguez-Clark https://crypto.interactive-maths.com/autokey-cipher.html

Testing:  
Running the Triple des brute force from the bottom up will converge after about two hours (on the computer it was ran on).

Discussion:

A few critiques can be raised for some of the implementations in the project. The first being that I was not able find a method to produce plaintext from the cipher in part 1 task 3. One could also critique the implemented function as not being a function which takes in ciphertext and produces plaintext, since it requires some human input in-between processes. However, the task was after all to produce tools to help produce the plaintext.

The TripleDes brute force also took quite a while, and it would be better suited to the theme of the project to find a faster solution, (to demonstrate that in fact this cipher also can be easily cracked). Since the slow solution gave the correct answer, I chose not to investigate it further although I will mention that one can use strategies such as multithreading /processing to speed up this process. In this specific case when the plaintext is pure letters, one can also choose to break the loop earlier if the key suggestions produce non-letter ascii symbols, this might speed up the process.

Conclusion:

In this project we were able to see how certain encryption schemes can be cracked. Statistical techniques can be used to crack polyalphabetic ciphers, and the SDES algorithm which models the larger DES is susceptible to brute-force attacks.