

NUMBER SEQUENCE

(A REALLY LONG ONE) - PART 3

Cryptography Project

Presented to
Prof. Auston Davis
Department of Computer Science
San José State University

In Partial Fulfillment
of the Requirements for the Class
FA 17: CS265 Sec 01

By
Kaushik Murli
Yuvraj Singh Kanwar
October 6, 2017

Table of Contents

Problem Statement.....	2
Selected Cryptographic System	2
Substitution Cipher	2
Software to Implement the System	3
Software to Implement an Attack on the System.....	4
Work Factor	5
Computational Analysis	6
Conclusion.....	6

Problem Statement

Substitution ciphers are the oldest form of ciphers ever developed. The project aims to use substitution cipher to encrypt text. We have the first 251 numbers of the sequence (a1, a2...a250, a251). Our aim is to find 100 numbers of this sequence starting with the 10001st number (a10001...a10100) and then apply modulo 7 operation to the resultant sequence. This reduced sequence is of the format (1,5,0,6...2) which is our key.

We extend from the scope of the challenge to apply the algorithm and encrypt the text by the numbers given in the sequence. The same is performed for decryption. For example,

If we have generated the sequence, 1,4,5,0,4

And we have plain text as HELLO

Then, we will get cypher text as IIQLS

Cipher Challenge:

NUMBER SEQUENCE (A REALLY LONG ONE) - PART 3

Author: Viktor Veselovsky (August 2011)

Selected Cryptographic System

Substitution Cipher

A cipher that has survived hundreds of years and is still being used is none other than substitution cipher (a detailed history and excellent description is given in 'the Code Book' written by Simon Singh). Basically, it constitutes substituting every element of the plaintext for a different character. It varies from the Caesar cipher in a way that the cipher alphabet is not shifted simply, it is permuted.

Very less communication security is offered by simple substitution cipher, and it can be easily broken even by hand, more so when the messages are longer.

Below is an example of cryptographic encryption and decryption steps that are involved with simple substitution cipher. The text we will encrypt is "A really long number sequence".

Keys for the simple substitution cipher mostly contain 26 letters of the English alphabet (as compared to single number in the Ceaser cipher). An example key is:

Plain text: abcdefghijklmnopqrstuvwxyz

key: phqgiumeaylnofdxjkrvstzwb

An example encryption using the above key:

Plaintext: A really long number sequence

Ciphertext: p kipyyw yfom ovlhik rixvioqi

It is easy to identify from the above example that each character in the plaintext is replaced with the corresponding letter in the key. Decryption is equally simple, by substituting alphabets from the cipher text back to the plain text.

In our project we are going to use a mathematical sequence to generate a complex key which will be used to substitute each letter of plain text to encrypt it. We created algorithm to find 100 numbers of this sequence starting with the 10001st number (a10001...a10100) of the given sequence of 251 numbers. After finding out the required sequence of number we reduce the sequence of numbers by applying modulo by 7, operation. This reduced the sequence to the format (1,5,0,6...2) without spaces. For example,

If we have generated the sequence, 1,4,5,0,4

And we have plain text as HELLO

Then, we will get cypher text as IIQLS

Software to Implement the System

Analyzing the given sequence, we observed a pattern where the difference between each terms follows { 4,3,3,4,3,3} and {5,6,7,5,5,8,6,5}. A number from the second array is inserted at every 4th and 6th term of the first array thus making the new array {4,3,3,4,5,3,3,6} to ensure randomness. Based on this pattern, we generate the 10000th to 10100 terms of the sequence.

We then apply modulo 7 operation on the above generated sequence to get a sequence that will be used as shift values for encrypting the text. An enhancement we applied is that when a space is encountered in the plain text, the space is consumed without using a key value, thereby getting a single word cipher text. This way the attacker cannot guess where the word starts and end thereby adding more work for the attacker. Another possible enhancement would be to use the key from various ranges between the 1st to the 10100th term to ensure randomness.

Our challenge only involves breaking the sequence, we then took this theoretical aspect and implemented this as our cipher.

For Example:

If we get the end sequence as

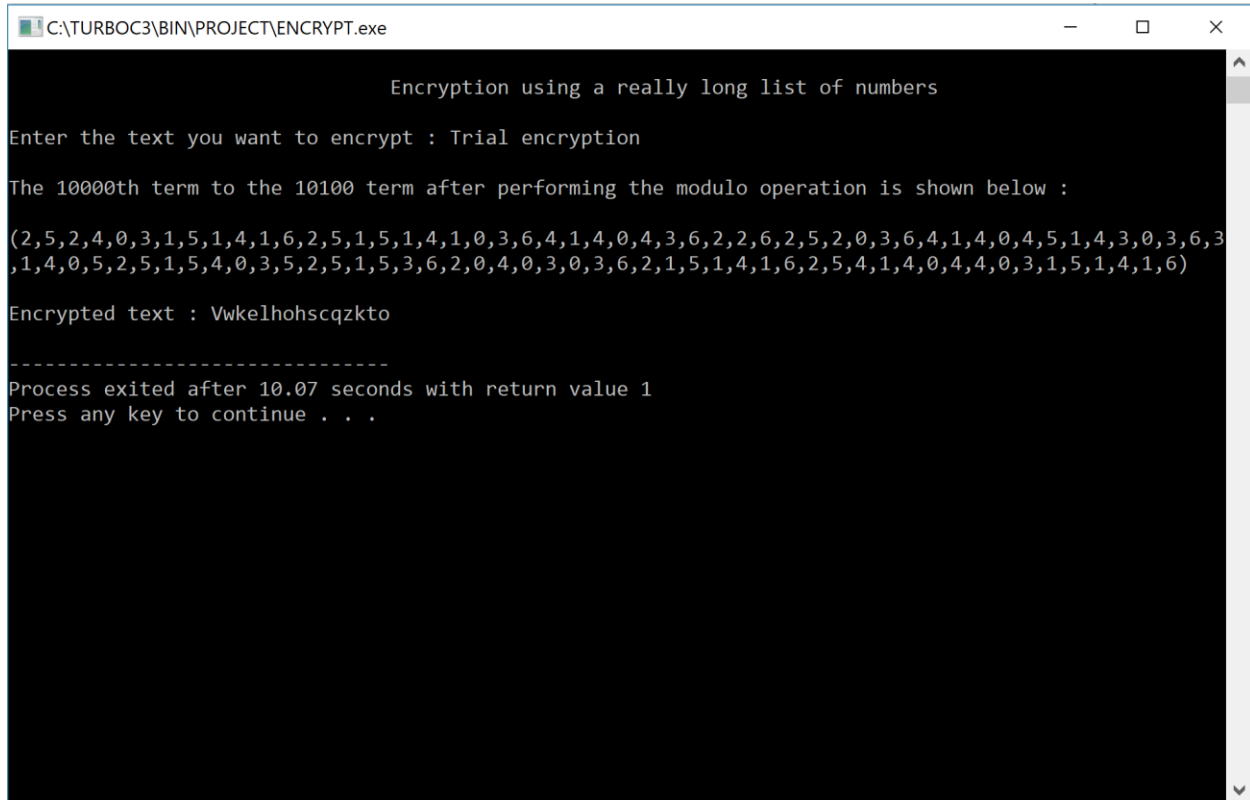
(0,4,0,3,0,3,6,3,6,2,6,2,5,2,5,1,5,1,4,1,4,0,4,0,3,0,3,6,3,6,2,6,2,5,2,5,1,5,1,4,1,4,0,4,0,3,0,3,6,3,6,2,6,2,5,2,5,1,5,1,4,1,4,0,4,0,3,0,3,6,3,6,2,6,2,5,2,5,1,5,1,4,1,4,0,4,0,3,0,3,6,3,6,2,6,2,5,2,5,1)

Name & Email id: Kaushik Murli (kaushikmurli@gmail.com), Yuvraj Kanwar (Yuvraj.kanwar@sjsu.edu)

Plain text: Trial encryption
Cipher text: Vwkelhohscqzkto

Source Code: Encrypt.cpp

Screen:



```
C:\TURBOC3\BIN\PROJECT\ENCRYPT.exe

Encryption using a really long list of numbers

Enter the text you want to encrypt : Trial encryption

The 10000th term to the 10100 term after performing the modulo operation is shown below :

(2,5,2,4,0,3,1,5,1,4,1,6,2,5,1,5,1,4,1,0,3,6,4,1,4,0,4,3,6,2,2,6,2,5,2,0,3,6,4,1,4,0,4,5,1,4,3,0,3,6,3,
1,4,0,5,2,5,1,5,4,0,3,5,2,5,1,5,3,6,2,0,4,0,3,0,3,6,2,1,5,1,4,1,6,2,5,4,1,4,0,4,4,0,3,1,5,1,4,1,6)

Encrypted text : Vwkelhohscqzkto

-----
Process exited after 10.07 seconds with return value 1
Press any key to continue . . .
```

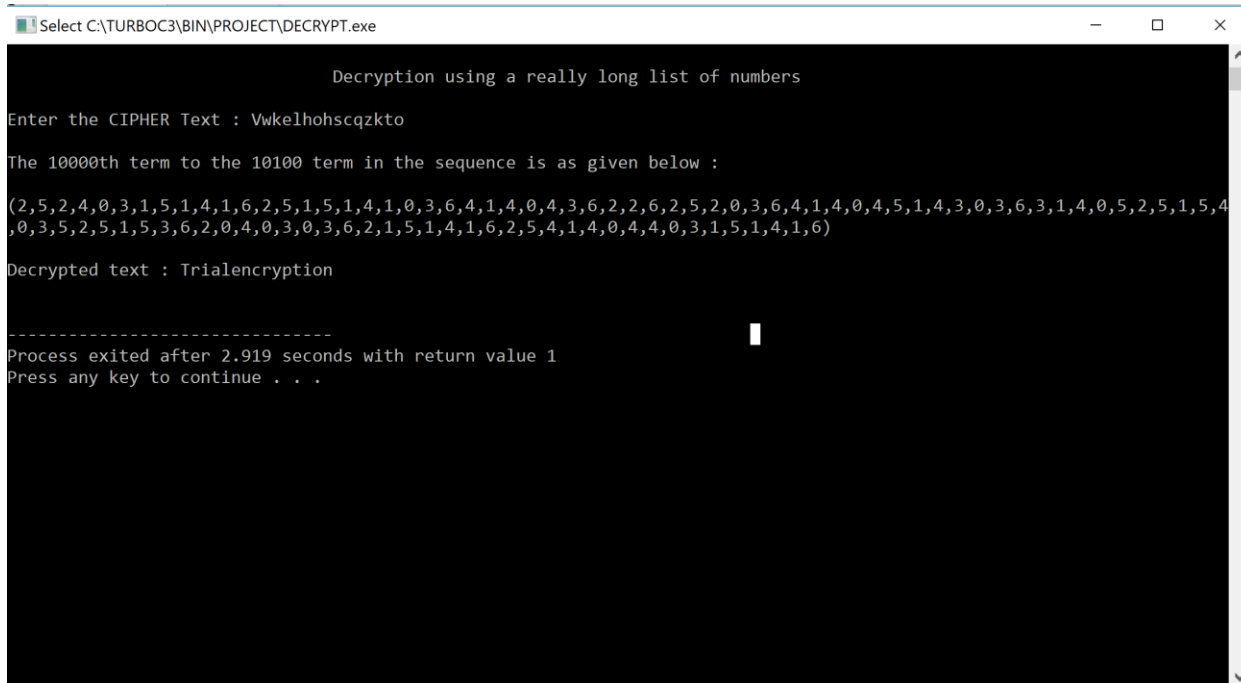
Software to Implement an Attack on the System

To decrypt our implementation, Trudy (hacker) will have to figure out that the characters have been shifted and that a substitution cipher is used. Trudy might be able to figure out that the maximum shifts occurring for characters are in the range of 0-6 and that modulo 7 operation has been performed. We assume that Trudy is aware of the workings of the algorithm and she manages to get a piece of the initial sequence say the first 30 terms. She can then figure out the difference as we did and generate the sequence for all the 10000 terms and then perform the modulo 7 operation. Now, brute force attack is necessary to figure out where the key starts from. Thus, Trudy has to decrypt the plain text starting from the 10000th term till 10097th term to figure out where the key starts. This will be an optimal attack for Trudy as 100 different executions isn't so bad.

Name & Email id: Kaushik Murli (kaushikmurli@gmail.com), Yuvraj Kanwar (Yuvraj.kanwar@sjsu.edu)

Decryption Text Entered: Vwkelhohscqzkto
Decrypted text: Trial encryption

Source: DECRYPT.CPP
Screen:



```
Select C:\TURBOC3\BIN\PROJECT\DECRYPT.exe

Decryption using a really long list of numbers

Enter the CIPHER Text : Vwkelhohscqzkto

The 10000th term to the 10100 term in the sequence is as given below :

(2,5,2,4,0,3,1,5,1,4,1,6,2,5,1,5,1,4,1,0,3,6,4,1,4,0,4,3,6,2,2,6,2,5,2,0,3,6,4,1,4,0,4,5,1,4,3,0,3,6,3,1,4,0,5,2,5,1,5,4,0,3,5,2,5,1,5,3,6,2,0,4,0,3,0,3,6,2,1,5,1,4,1,6,2,5,4,1,4,0,4,4,0,3,1,5,1,4,1,6)

Decrypted text : Trial encryption

-----
Process exited after 2.919 seconds with return value 1
Press any key to continue . . .
```

Other possible attacks would be:

- Based on the spacing of each word, Trudy can pick a small one and try to work out the difference pattern in this.

Ex: Cipher text: Xnb shdajdy

Xnb could represent the.

But this would require a lot of complex computations. This will not work on our implementation as we have consumed the space from the cipher text thereby averting this attack.

- Trudy can attempt to reverse the modulo 7 operation to get the sequence of 10000th term to 10100 terms to observe the pattern. Pattern of the sequence if identified by Trudy can be coded to decrypt the text.

Work Factor

Encryption process: 8 secs (Depending upon input)

1. Generation of Sequence and modulo operation: 3 secs.
2. Substituting characters based on generated sequence: 4.12 secs.

Name & Email id: Kaushik Murli (kaushikmurli@gmail.com), Yuvraj Kanwar (Yuvraj.kanwar@sjsu.edu)

Total: ~ 8 secs – 10 secs based on input.

Decryption process: 7 secs (Depending upon input)

1. Generation of Sequence and modulo operation: 4.472 secs.
2. Substituting characters from cipher text to plain text based on generated sequence: 3 secs.

Total: ~ 7 secs – 8 secs based on input.

Computational Analysis

A time consuming process was to figure out the pattern present in the sequence. A number of methods were test initially but majority of them failed. Some of them are:

- Arithmetic progression
- Geometric progression
- Recurring relations
- Diophantine Equation

We found some success when we calculated the difference between the successive terms but even they did not have an apparent pattern. We initially found a sequence of 4, 3, 3, 4, *, 3, 3, *. The 4,3,3,4 pattern and the 3, 3 pattern kept repeating but we could not figure out the * terms. After much head scratching, we found a pattern after isolating the * terms from the rest of the sequence. Another subsequent pattern was found in the * terms.

We then coded the algorithm and generated the sequence which was used to encrypt the text. The encrypted text was then fed into our decryption algorithm and obtained the original plain text thereby verifying our analysis.

Conclusion

This substitution cipher employs a considerable amount of confusion and minimal diffusion to encrypt text in an efficient manner. Repetitive words like the... does not result in the same cipher text as the sequence keeps varying. This reduces the chance of frequency search and forward search attacks thus giving us a strong cipher.

But this cipher tends to be weak if someone figures out the pattern to the sequence as shown in our attack.