# Development of simple and relatively strong encryption algorithm using combination of XOR and bit shifting

Author: Lisnic Andrei, student of Technical University of Moldova, gr. FAF-081

andrei.lisnic@gmail.com

***Abstract – this article will describe how to easily develop a efficient encryption algorithm that is obtained by combination of simple encryption methods. This Combination will protect encrypted data from being analyzed semantically.***

**Introduction**

Cryptography is the art and science of secret writing. The term is derived from the Greek language

- *krytos* - secret
- *graphos* - writing

Encryption is the actual process of applying cryptography. Much of cryptography is math oriented and uses patterns and algorithms to encrypt messages, text, words, signals and other forms of communication.
Cryptography has many uses, especially in the areas of espionage, intelligence and military operations. Today, many security systems and companies use cryptography to transfer information over the Internet or radio for fears of interception. Some of this encryption is highly advanced, however even simple encryption techniques can help uphold the privacy of any everyday person.
The term cryptography also meant the breaking of encrypted messages until the early 1920s, when the concept of *Cryptanalysis* began being used and is now practically an art and science all on its own.
The two main areas of cryptography are *Cipher* and *Code*.

**Developing the algorithm**

My work course was to develop a program that can encrypt and decrypt data using keys, to develop such a program I set some objectives:

- encryption key must be inputed from user, and should be any combination of characters/symbols.
- For 2 different keys the program must generate 2 different sets of data.
- Data decrypted with a wrong key must not be similar to data encrypted with correct key.
- The encrypted data must not be analogical to original data in sense of semantic analysis.

Usually the XOR operation is used to encrypt the data. This is an extremely insecure algorithm but despite this it is relatively widely used.
I decided to use a Vigenere type cipher algorithm. It is often used for low-security applications. The first byte of the file is encrypted with the first character of the password, the second byte with the second character, and so on. If all the characters of the password have been used, the next byte of the file is encrypted with the first character again.
To encrypt the byte with the character typically the XOR operation is used. This operation has the property that if you apply it twice with the same character, you get the original byte back. This makes it extremely easy to implement encryption and decryption.
The Vigenere algorithm is very insecure. Encrypted data can be easily analyzed semantically.
With many types of files, the first few bytes are always the same so
that the operating system can tell what type of file it is. GIF images for
example start with "GIF87" and Word documents start with "MSW". This is very
convenient for breaking the code, since we know a part of original data.

Additionally, passwords often consist of letters in all lower case and so do many files that are encrypted this way (because they are text documents). This results in patterns that are easily recognizable in the encrypted file.

So, to add some security to my algorithm, I added bit shifting to it. A part of encrypted byte will be shifted to next byte that will be encrypted. Thanks to that, a part of

all data will be encrypted twice, since it's being shifted before encryption.
The amount of shifted bits must be dynamic and must be generated by the key.

So all the algorithm can be summarized to the following 2 steps:

```
1. Put shifted bits from previous byte
in CurrentByte and shift CurrentByte
and memorize the lost part ( it will be
put in the beginning of next byte)
2. CurrentByte XOR ( Key[index] OR
KeyChecksum)
```

At this moment the algorithm is done, next step is implementation, which is not so difficult.

**Algorithm visualization**

I will schematically visualize how the chosen encryption method works on a byte from the original data.

| Current byte | Current key | Key Checksum | Lost bits | Resulting byte |
|---|---|---|---|---|
| **Initial data** | | | | |
| 10011011 | 1100101 | 00110010 | 01000000 | none |
| **Stage 1** | | | | |
| 01100110 | 1100101 | 00110010 | 11000000 | none |
| **Stage 2** | | | | |
| 01100110 | 1100101 | 00110010 | 11000000 | **10011001** |

**Cryptographic analysis**

Here is how encrypted data behave in different situations:
Initial data:

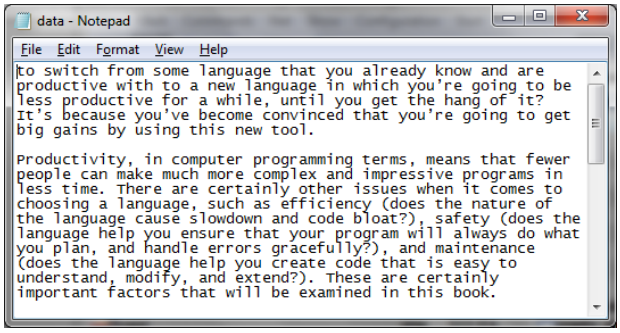


Fig. 1: original data

The following file is the result of encryption of original data with the key “conference”:

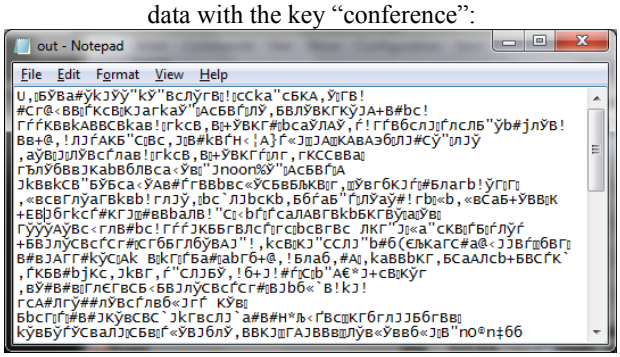


Fig. 2: encrypted data

The following chart represents the relative correlation of initial data contents:

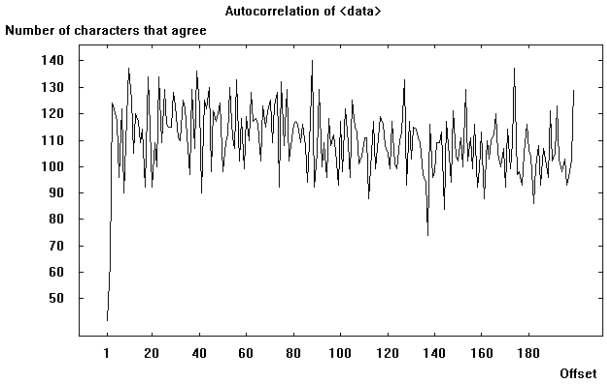


Fig3: autocorrelation of initial data

You can see that the relative correlation is different in the encrypted file, which is a very nice result, meaning that we have different semantical repartition:

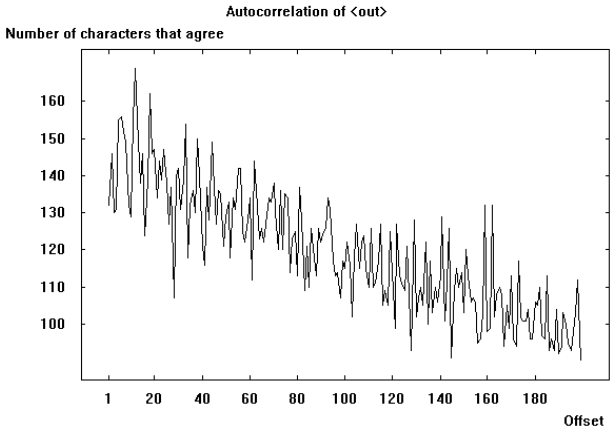


Fig. 4: Autocorrelation of encrypted data

Besides autocorrelation, the floating frequency must be analysed, it represents how diverse the data become after encryption:

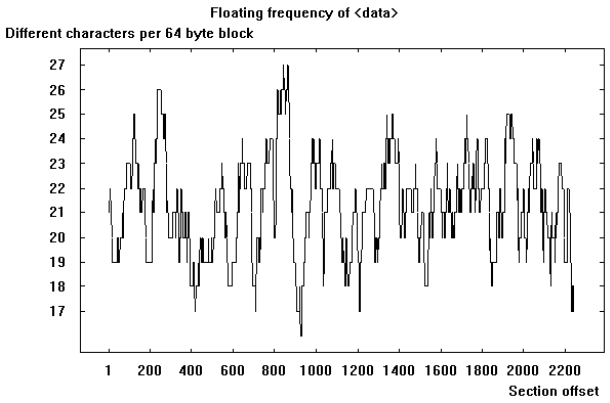


Fig. 5: Floating frequency of initial data

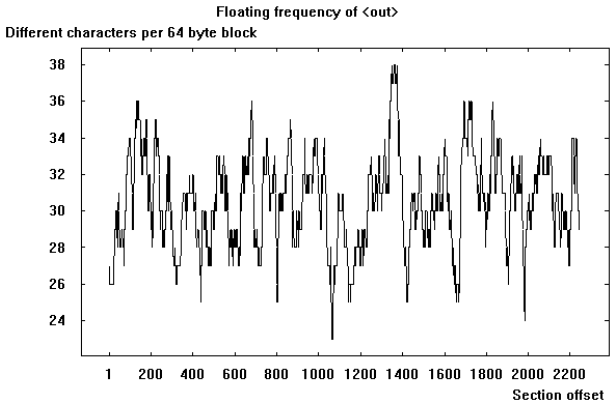


Fig. 6: Floating frequency of encrypted data

At this moment it is clear that the data was not only **ciphered**, but also **coded,** so practically it's impossible to analyze semantically the encrypted file, because it has another data frequency.

But besides that, the algorithm must be checked for behavior with similar keys. Here is the result of decryption of the encrypted file with wrong key **conferencf** (1 character difference):

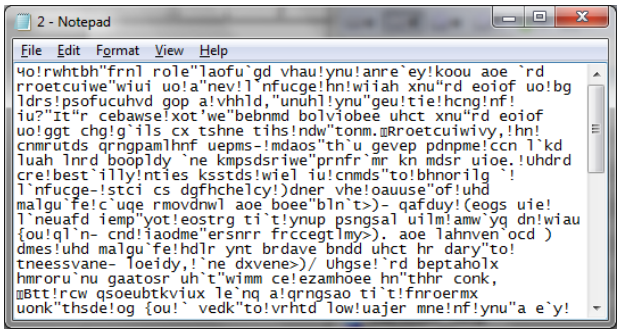


Fig. 7: the file decrypted with similar, but wrong key
Here is the result of similarity analysis:

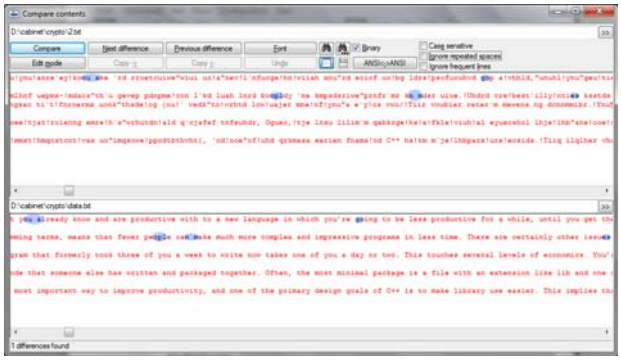

Fig. 8: Similarity between original file and the one decrypted with wrong key

**Conclusion:**
With the difference of only 1 character, the algorithm generate almost different files, considering the simplicity of the program, that's a very satisfying result.