EsoCipher: An Instance of Hybrid Cryptography

Neel Adwani
University of Petroleum and Energy Studies
Dehradun, Uttarakhand
contact@neeltron.com

ABSTRACT
This paper proposes a whole new concept in the field of Cryptography, i.e., EsoCiphers. Short for Esoteric Cipher, EsoCipher is an algorithm, which can be understood by a few, who have the knowledge about its backend architecture. The idea behind this concept is derived from esoteric programming languages, but EsoCiphers will be having a practical use in the future, as more research is done on the topic. Although the algorithm is quite simple to understand, the complexity of the output will indeed prove to be difficult to brute force if a secret is embedded to it. It uses a hybrid cryptography based technique, which combines ASCII, Binary, Octal and ROT 13 ciphers. The implementation and similarity index has been provided to show that it can be implemented practically.

1 INTRODUCTION
According to the international standards, the term "esoteric" is defined as something that can be understood by a small group of people, having knowledge of the same. A common term, similar to it is "esolang", short for esoteric programming language [10] which can be further stated as a language that can be decoded by some specific compilers or interpreters. One such example of an esoteric programming language is Malbolge [7]. As for Esoteric Cipher, it is not entirely an esoteric language, nor an ordinary cipher, but a mixture of both of them, making it entirely a hybrid cryptographic technique.

In Layman terms, cryptography is a study of different techniques, in order to protect data or any sort of communications happening through any medium. A Cipher would be defined as an algorithm that can encrypt or decrypt the data. In our case, the algorithm is totally dependent upon communication of text based messages, and the key to it will be as per the user’s preference. Initially, the user will have to type the message, and then the secret. The secret will be embedded to the message, which is similar to salting but not exactly the same. Upon execution of the algorithm, it will translate the message into an entirely tangled string, which would be almost impossible to decipher without having the secret.

* The terms key and secret can be used interchangeably, within the scope of this paper.

2 RELATED WORK
This section demonstrates a glimpse of related algorithms that have been implemented and are currently in use.
Malbolge [7] is an esoteric programming language, which uses characters based on the ASCII table and has a specified range of printable and non-printable characters, depending upon that range. Instead of Deciphering algorithms, esoteric languages have interpreters and they’re mostly proposed as jokes. Another popular esoteric language is Brainfuck [5], which became notable for its minimalistic expressions. The whole language constitutes 8 simple commands and an instruction pointer. LOLCODE [8] is yet another esoteric language, which is based on the lolcat meme. So, its files are generated of the extensions .lol and .lols. Since its inspiration comes from memes over the internet, the syntax contains keywords like HAI, BTW, GTFO, KTHXBYE.

While there’s no intended practical use of esoteric languages, EsoCipher may be considered obfuscated but it can be proven useful in some fields. Since a lot of exploration hasn’t been done around this topic, it is hard to determine the usability of the given algorithms in this paper, but intentionally EsoCiphers can be useful in preventing the attacker from reading the data during data breaches. It can even be used in Machine Learning/Deep Learning Models where sensitive data is required to train the model. No human will be able to make sense out of that data, without knowing the key and the exact functions used in their algorithm. On another note, the given algorithms can be modified, depending upon the requirements of the developer.

3 ALGORITHMIC AND PROCEDURAL BACKGROUND
This section presents the back-end of this algorithm, which provides a good idea about its working approach and the ways it can be implemented in different languages.

3.1 Encipher
During the process of enciphering, the algorithm initially replaces special characters like ‘;’, ‘@’, ‘!', etc., with their alphabetical counterparts like ‘period’, ‘comma’, ‘at’, ‘exclamation’, which are absolutely flexible, making the cipher flexible and secure, as per the special characters are defined. Moving forward, the ASCII [3] value of each alphabet is converted into binary and then embedded into a new variable as a string. The character ‘b’ has to be removed as it may cause conversion errors. After that, the octal [6] value of that string is taken into account for an integer of the whole string and finally, the ROT13 [4] Algorithm is executed and the enciphered string obtained by the EsoCipher Algorithm is obtained. A summarized visualization for the same is shown in Fig. 1.

3.2 Decipher
In order to decipher the EsoCipher, the program needs an ‘EsoCiphered’ string to begin with. After the successful input of the string, it is looped through and a ROT -13 is executed on it, i.e., the ASCII value of each character is decreased by 13 and stored collectively in a variable. For each group of 8 characters, a 0 is added initially and then an octal value is received. The octal value is then taken into account and converted to binary. For each series of 8, the binary
Algorithm 1: Generating an EsoCipher

Result: The EsoCipher is returned as a string.

\[
\text{input\_string} = \text{""}, \text{modified\_string} = \text{""}, \text{iter} = 0, x = 0, \\
\text{temp\_string} = \text{""};
\]
\[
\text{input\_string} \leftarrow \text{"SampleString"};
\]
\[
\text{while iter < len(input\_string)} \text{ do}
\]
\[
\begin{align*}
x & \leftarrow \text{binary}(\text{ascii}(\text{input\_string}[\text{iter}])); \\
\text{modified\_string} & \leftarrow \text{modified\_string} + \text{string}(x);
\end{align*}
\]
\[
\text{end}
\]
\[
\text{modified\_string} \leftarrow \text{modified\_string}.\text{replace}(\text{"'b"}, 1);
\]
\[
\text{temp\_string} \leftarrow \text{octal}(\text{int}(\text{modified\_string}));
\]
\[
\text{modified\_string} = \text{""};
\]
\[
\text{iter} = 0;
\]
\[
\text{while iter < len(temp\_string)} \text{ do}
\]
\[
\begin{align*}
x & \leftarrow \text{ascii}(\text{temp\_string}[\text{iter}]); \\
x & \leftarrow x + 13; \\
\text{modified\_string} & \leftarrow \text{modified\_string} + \text{alpha}(x);
\end{align*}
\]
\[
\text{end}
\]
\[
\text{return modified\_string};
\]

Algorithm 2: Deciphering the EsoCipher string

Result: The EsoCipher is deciphered and returned as a string.

\[
\text{esocipher} = \text{""}, \text{deciphered\_string} = \text{""}, \text{iter} = 0, x = 0, \\
\text{temp\_string} = \text{""};
\]
\[
\text{while iter < len(esocipher)} \text{ do}
\]
\[
\begin{align*}
x & \leftarrow \text{ascii}(\text{esocipher}[\text{iter}]); \\
x & \leftarrow x - 13; \\
\text{deciphered\_string} & \leftarrow \text{deciphered\_string} + \text{char}(x);
\end{align*}
\]
\[
\text{end}
\]
\[
\text{deciphered\_string} \leftarrow \text{"0" + string}(\text{int}(\text{deciphered\_string}, 8));
\]
\[
\text{temp\_string} \leftarrow \text{octal}(\text{int}(\text{modified\_string}));
\]
\[
\text{for iter in modified\_string[::8]} \text{ do}
\]
\[
\begin{align*}
\text{final\_string} & \leftarrow \text{final\_string} + \text{char}(\text{iter});
\end{align*}
\]
\[
\text{end}
\]
\[
\text{return modified\_string};
\]

Figure 1: Flowchart for understanding the Esocipher

Figure 4: Implementation of EsoCipher

This section demonstrates an in-depth explanation of the way this algorithm was implemented in Python 3.7 and it includes relevant screenshots attached to it.

For the enciphering algorithm, no external libraries are needed to be imported, unless a different cryptic function has to be used. As for the algorithm, the users are allowed to set their own codes to punctuation marks, so basically for more number of punctuation marks, the cipher will get stronger. A string “hello world” is supplied as an input to our program, with the notation for ‘ ‘ as “space_bar”. The output turned out to be as shown in Figure 3.

For Deciphering of the same generated string in Figure 3, a separate algorithm was programmed in the same development environment, as per Algorithm 2, stated above in this paper. The string that is supposed to be inputted is the one generated in Figure 3. For looking into the results, refer to Figure 4.

5 SIMILARITY INDEX AND LOSSES

This section is about testing Algorithm 1 on a random paragraph, Algorithm 2 on the result generated from Algorithm 1, and calculating the Levenshtein Distance [2] [9] between the original string (random paragraph) and the output of Algorithm 2. Since the punctuation marks are to be defined by the user, it is assumed within the scope of this paper, that all the punctuation marks in the paragraph are annotated in both the algorithms.

A short excerpt from a Harry Potter novel is taken, which went through both the algorithms and the Levenshtein distance between
EsoCipher: An Instance of Hybrid Cryptography

Figure 2: Flowchart for understanding the process of deciphering an Esocipher

Figure 3: Output during Enciphering

Figure 4: Output during Deciphering

Figure 5: Implementation of Levenshtein’s Distance Algorithm

and even 0 if all the punctuation marks are defined correctly in the source code.

REFERENCES