Seven Steps to a Forever-Safe Cipher

(An Introduction to Poly-Substitution Encryption)

Julián Murguía Hughes
Independent Researcher
Montevideo, Uruguay
email: jmurguia@montevideo.com.uy

Abstract—**All cryptography currently in use is vulnerable to an attacker with enough computational power and most of them will become obsolete once quantum computing becomes widely available. Continuing the current path seeking for more and more complex algorithms cannot guarantee neither secrecy nor unbreakability. Increasing the complexity while it keeps being vulnerable does not seem to be the right approach. Thinking outside the box is not enough. We need to start looking from a different perspective for a different path to ensure data privacy and secrecy. In this paper, we introduce Poly-Substitution encryption and share advances in searching for unconditional security instead of complexity and we try to light a path to a whole different cryptography based on simplicity and resistant not only to quantum attacks but also to what may come later, including attackers with infinite computational power.**

Keywords-cipher; poly-substitution; unconditional security; perfect secrecy; inifinite computational power; quantum-resistant; cryptography; secrecy; unbreakability; privacy; encryption; quantum; computing; resistant; data.

I. INTRODUCTION

In this work in progress, we show current achievements in the field of cryptography and present some future ideas in this area and their potential. No final results or final data is available at this time.

This work updates, continues and expands our paper "Seven Steps to a Quantum-Resistant Cipher" presented at SECURWARE 2016, The Tenth International Conference on Emerging Security Information, Systems and Technologies; held in Nice, France – July 24-28, 2016 [1].

Since the beginning, cryptography has worked the same way; you take the original source of information (the plaintext), a key and a fixed substitution and you apply the substitution using the plaintext and the key as input to generate the cryptogram or ciphertext as its output. Modern cryptography keeps working in the exact same way.

A. Definitions

To set a common ground and avoid confusions and misunderstandings, a minimum set of definitions is required and listed here:

Symbol. A symbol is a representation of something. From a single character in any given language like English or an ideogram in Chinese to an abstract concept like π representing the relation between a circumference and its diameter, which is a numeric value with infinite decimal values never repeating.

Alphabet. An alphabet is a finite set of symbols listed in a given order.

Shifted alphabet. It is an alphabet where the symbols are shifted place by a given number of positions from the original order and the alphabet is considered circular for the shifting process, where the first symbol follows the last one and the last symbol precedes the first one.

Mixed or Permuted alphabet. It is an alphabet where the order of the symbols is arbitrarily mixed or permuted from the original order.

Word. A word is a finite sequence of symbols in an arbitrary order where not all the symbols from the alphabet need to be present and any symbol may appear more than once. The meaning of a word does not depend on the order of the symbols within the alphabet.

Phrase. A phrase is a finite sequence of separated words.

Text. A text is a finite sequence of separated phrases.

Dictionary. A dictionary is a text listing all valid words and using phrases to define the meaning of each one.

Plaintext. The original unencrypted text or message.

Ciphertext. The result of encrypting the plaintext.

Unconditionally Secure System. We will use the definition given by Whitfield Diffie and Martin E. Hellman [2] as they stated that "a system that can resist any cryptanalytic attack, no matter how much computation is allowed, is called unconditionally secure".

B. Caesar Cipher

Although the first known evidence of some form of cryptography is almost four millennia old [3], one of the oldest known forms of encryption is the Caesar's cipher. It was a substitution cipher where each character was replaced for the one located three places later in alphabetic order and considered the alphabet as a round circle as it is shown in Figure 1, where 'A' follows 'Z' and so, 'X' would be replaced by 'A', 'Y' would be replaced by 'B', 'Z' would be replaced by 'C', 'A' would be replaced by 'D' and so on.

To encrypt or cipher a plaintext using Caesar's cipher, each character from the plaintext is replaced by the one placed three positions moving clockwise. To decrypt or decipher a ciphertext, each character from the ciphertext is replaced by the one located three positions moving counter clockwise.

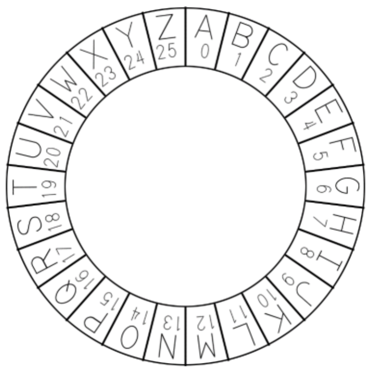

Figure 1. Circular positional alphabet and position values.

The Caesar's algorithm was just a shift by places process and the key used was just three, indicating the algorithm that each character in the plaintext needed to be shifted by three to generate the cryptogram. All shift by places encryption algorithms are generically referred as Caesar ciphers. As in this type of cipher each letter is replaced always by the same letter, it is called a mono-alphabetic substitution cipher.

The Caesar's cipher can be represented using modular arithmetic. Modular arithmetic is a system of arithmetic for integer numbers where values wrap around upon reaching a maximum value.

To represent Caesar's cipher using modular arithmetic, we start by assigning a numeric value to each letter from the alphabet according to their position within such alphabet. In the classic English alphabet and its standard alphabetic order, to the letter "A" corresponds the value 0 (zero), to the letter "B" corresponds the value 1, and so up to the letter "Z" with a value of 25. Figure 2 shows a traditional positional alphabet and the numeric value associated to each letter of such alphabet.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Figure 2. Positional Alphabet and Position Values.

As the maximum value is 25 and the value 26 wraps around to zero, the modulus for the Caesar's cipher will be 26.

The substitution of any letter (x) for the one located *n* places to the right can be represented through the mathematical formula:

$$E_n(x) = (x + n) \bmod 26 \quad (1)$$

For the original Caesar's cipher, the formula to cipher would be:

$$E_3(x) = (x + 3) \bmod 26 \quad (2)$$

The reverse deciphering process can be represented through the mathematical formula:

$$D_n(x) = (x - n) \bmod 26 \quad (3)$$

For the original Caesar's cipher, the formula would be:

$$D_3(x) = (x - 3) \bmod 26 \quad (4)$$

Although it is considered obsolete and today it can be broken without the need of a computer, just with pencil, paper and some spare time, it lasted for centuries.

C. Vigenère Cipher

In 1553, Italian cryptologyst Giovan Battista Belasso described in his book [4] a new cipher later attributed to Blaise de Vigenère and which is still known as the Vigenère cipher.

This cipher, instead of using a single key value for the substitution, uses a sequence of letters so that instead of performing a mono-alphabetic substitution, performs what is called a variable or poly alphabetic substitution, where each letter may produce a different result.

Vigenère's encryption is functionally based on the use of the tabula recta, invented by German monk Johannes Trithemius in 1508, which is a square table of alphabets where each row is made by shifting the row above one position to the left.

To cipher, the plaintext character to be encrypted is looked into the table's first row, the character from the key to be used is looked into the table's first column and the ciphertext character will be the one located at the intersection of the column corresponding to the plaintext character with the row corresponding to the key character. To decipher, the key character is looked into the first column and then that row is looked for the position of the ciphertext character. Once found, the character at the top of this column will be the plaintext character.

Figure 3 shows the Tabula Recta created by Johannes Trithemius, which is also called the Vigenère table.

A	B	C	D	E	F	G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 3. Vigenère Table.

The Vigenère cipher can also be represented using modular arithmetic, assigning a value to each letter, the same way we did with the Caesar's cipher in Figure 1 and also in Figure 2.

To cipher, for each letter in the plaintext message and the corresponding key letter, the alphabet position values are added using modular addition module 26 and the resulting value will indicate the position of the ciphertext letter corresponding to the result.

Being *x* the letter in position *n* within the plaintext message we want to cipher and *k* the corresponding letter from the key to be used, the cipher process can be represented using the following formula:

$$E\ (x_n) = (x_n + k_n) \bmod 26 \qquad (5)$$

To decipher, the formula would be:

$$D\ (x_n) = (x_n - k_n) \bmod 26 \qquad (6)$$

The Vigenère cipher is known for being easy to be understood and implemented and hard to break. It lasted for almost four centuries, as we will see when we address its vulnerability in Section II.

D. Vernam Cipher

About a century ago, Gilbert Vernam invented an encryption technique [5] (Patent US 1310719 [6]) that thirty-something years later Claude Shannon proved [7] it offered Perfect Secrecy and properly used will remain impervious to any attack no matter how powerful the attacker may be, including quantum computing and even an attacker with infinite computational power. It is not used because it requires the key to have the same length as the plaintext, to be truly random and not to be reused. Those constraints were considered and are still considered strong enough to prevent its usage.

As today's information is always measured in bytes or multiples of byte sizes (Kilobytes, Megabytes, Gigabytes, Terabytes, etc.) for all the explanations and examples here, the byte as the basic unit of information will be used. Considering the byte as just a group of eight bits, being a bit a binary digit that can either have a value of zero (0) or one (1).

A single byte can represent 256 different values, from 0 to 255 in decimal notation, from 00 to FF in hexadecimal notation and from 00000000 to 11111111 in binary format representation.

For a byte, the Vernam cipher will perform exactly the same way as for a single bit, it will use the XOR function between the plaintext byte and the key byte. The behavior of the function is simple, it will compare each bit within the byte from the plaintext to the bit in the same position in the byte from the key and will generate a bit with a value of zero if both bits have the same value and one if they are different. This XOR function will return the cryptogram or ciphertext byte as its result. For a specific plaintext byte value, each of the 256 possible values of the key byte will produce a different ciphertext byte value.

If you get the cryptogram or ciphertext byte and do not know the value of the key byte, every single possible value of the key byte has the exact same probability of being the right one and you have no way to decide which one of them is the right one and thus, which of the 256 possible values of the plaintext byte is the right one.

There is no possible cryptanalysis of this process and a brute force attack will end up with the plaintext mixed with a huge number of false positives (apparently valid results that are not the original plaintext) with no way to tell which one is the original one.

Shannon proved that even knowing that the plaintext is just text, any possible text with the same length has the exact same probability of being the original plaintext [8].

Since then, algorithms have grown in complexity looking to enhance the security of the process and to make harder to recover the plaintext without knowing the key.

But what has not changed is the logic, i.e., the way it is done. Cryptography is still using an algorithm with a fixed set of instructions that will use the plaintext and the key as input to produce the ciphertext. The same plaintext and the same key will always produce the same cryptogram.

There are two main attacks to try to get the plaintext without knowing the key: Cryptanalysis (analyze the process trying to find weaknesses or shortcuts that may allow to retrieve the original information without having the key) and Brute Force (try all possible keys).

Modern cryptography is not unbreakable and bases its security on two premises:

1) Cryptanalysis is not possible or too complex to be achieved.
2) Brute Force attacks require too much time and computational power.

In this paper, we will prove that Caesar and Vernam ciphers are just reduced or limited versions of the Vigenère cipher; we will introduce our proposed poly-substitution encryption technique and the seven steps to build cryptographic algorithms based on it and also prove that the Vigenère cipher is a mono substitution cipher and a reduced or limited version of our proposed encryption.

The rest of this paper is organized as follows. Section II will analyze the Vigenère cipher and prove Caesar and Vernam ciphers are just limited or reduced versions of it and will also explain why we consider the Vigenère cipher as a poly alphabetic mono substitution cipher. Section III describes data persistence, cryptography state of the art and their vulnerability to quantum attacks. Section IV introduces poly-substitution encryption, its theory, basis, definitions and how it works. Section V describes each of the seven steps we defined to build a poly-substitution cipher based on our proposed encryption technique. Section VI analyzes a cipher constructed using our proposed encryption and presents an example of it and its results. Section VII compares this sample cipher against Vigenère, Vernam and other current standards. Section VIII analyzes all possible attacks to our proposed encryption. Section IX describes how partial data universes are handled by symmetric and public key cryptography and their limitations to offer Format Preserving Encryption (FPE); it will also show how our proposed technique handles them better. Section X describes how our proposed poly-substitution encryption can offer and provide unconditional security. Section XI addresses practical applicability. In Section XII we address key and message distribution taking advantage of the use of internet. Section XIII describes the conclusions and Section XIV describes the future work and goals.

II. They are all Vigenère

A. *Caesar is Vigenère*

It is trivial to prove that any generic Caesar cipher is a reduced or limited version of the Vigenère cipher, where the key is just one symbol or character long.

B. *Vernam is Vigenère*

At the level of one bit, modular addition module 2, modular subtraction module 2 and XOR, all behave the same way and are in fact the exact same operation as it is shown in Figure 4.

(p + k) mod 2		k	
		0	1
p	0	0	1
	1	1	0

(p - k) mod 2		k	
		0	1
p	0	0	1
	1	1	0

p XOR k		k	
		0	1
p	0	0	1
	1	1	0

Figure 4. One-Bit Binary Operations.

Based on that, it is trivial to prove that the Vernam cipher is a reduced or limited version of the Vigenère cipher, where the alphabet has only two different symbols or characters.

C. *Vulnerability of the Vigenère Cipher*

In 1863, Friedrich Wilhelm Kasiski published a book about cryptography [9], where he described a method for cryptanalysis or cryptographic attack based on the existence of repeated sequences within the ciphertext. He assumed those repetitions were caused by a key shorter than the message and that they represented repeated sequences within the plaintext encrypted using the same portion of the key.

The Kasiski method argued that the distance between repeated sequences was a multiple of the length of the key. Starting from that, searched for multiple repeated sequences, measured the distances between repetitions and calculated the greatest common divisor to find a value that will be the length of the key or a multiple of it. Once the length of the key is obtained, the ciphertext is divided into blocks of that size and sub-cryptograms are formed by taking the first character of each block, then the second one and so on. Each one of those sub-cryptograms will have been encrypted using the same symbol or letter and so each one of them will be a mono-alphabetic substitution and so we would be able to perform a simple statistical frequency analysis attack.

The Kasiski statistical frequency analysis attack is based on two pillars:

- The known statistical distribution of the letters in a regular text.
- The known distance between the letters from the alphabet.

The most common letters in English are letter “E”, letter “T” and letter “A”, in that order; and is known that letter “E” is the fifth letter of the alphabet and its value is 4, the letter “T” is in position 19, 15 positions to the right of letter “E”, and letter “A” is in position 0, 7 positions to the right of letter “T”.

The Kasiski statistical frequency analysis attack will search in each sub-cryptogram the frequency distribution of the encrypted letters, focusing on those with highest frequencies (those who should correspond to the letters “E”, “T” and “A”) and that also comply with the alphabet structure and the distance between the most common letters within the alphabet. As letter “E” is in position 4, the following formula is true:

$$\text{Key} = \text{cipher letter} - E = \text{cipher letter} - 4 \tag{7}$$

So, the relative position of letter “E” on each sub-cryptogram will form the key ciphered through a substitution cipher like Caesar’s with a displacement of 4 positions to the right. From it, the key could be retrieved by a simple Caesar decryption using a displacement value of 4.

Statistical analysis is also used to search for repeated n-grams (known sequences of letters, “e.g.”, bigrams “TH”, “HE”, “IN”; trigrams “THE”, “AND”, and so on).

The use of a mixed or permuted alphabet only adds some extra work but it is still vulnerable.

Christopher Swenson, on his book [10], explains the Index of Coincidence (IC) defined by William F. Friedman as a measure of how evenly distributed the character sequences are within the frequency distribution table.

He considered "The Complete Works of William Shakespeare" as an adequate representation of the English language and calculated its IC to be approximately 0.0639.

He defines theoretically perfect IC as if all characters occurred the exact same number of times so that none was more likely than any other to be repeated, so, for an alphabet of 26 characters, he calculated it to be 1/26, which is approximately 0.03846.

IC can also be used with bigrams (sets of two characters) and trigrams (sets of three characters) to measure how evenly distributed they are within their corresponding distribution tables.

The theoretically perfect IC for bigrams is approximately 0.0015 (1/26*26) and for trigrams is approximately 0.00006 (1/26*26*26).

In English, a maximum of 676 bigrams and 17,576 trigrams can exist, although not every one of them may be valid.

The three most common bigrams HE, TH and IN have an IC of about 0.035, 0.034 and 0.0189 respectively. The most common trigram THE, has an IC of about 0.022.

William F. Friedman [11] presents a practical example using IC to break a poly alphabetic mono substitution encryption when the key has been reused.

The conditions set by Shannon to the Vernam cipher for it to offer unconditional security (The key should be as long as the plaintext, it should be random and it should not be used again) makes it a One-Time-Pad and the same applies to the Vigenère cipher. A plaintext encrypted using the Vigenère cipher using a random key with the same length as the plaintext that is not used again offers the same unconditional security defined by Shannon.

D. Mono Substitution

Although being the Vigenère cipher a poly alphabetic one and considering it can be used in reverse and use the decryption process to encrypt and the encryption process to retrieve the original plaintext, the substitution used along the encryption or decryption processes is always the same on each instance.

Each symbol or character is processed using the exact same substitution, modular addition for the encryption and modular subtraction for the decryption.

That is the reason why we call the Vigenère cipher a poly alphabetic mono substitution cipher using mono substitution encryption.

III. State of the Art

A. Information and Data Persistance

Something that is not directly related to cryptography but needs to be considered together because it has a direct impact on the information life cycle is the persistence of any data or information digitally transmitted or stored. Any information digitally transmitted or stored, persists.

Transmitted data leaves traces and copies between the source and the destination. Even encrypted information, transmitted through secure connections travels from router to router, from server to server from the point of origin to the destination point, and it can be copied in travel without being noticed.

Stored data also leaves copies behind. To totally delete specific data is very but very hard and cannot be assured nor guaranteed. Computer forensic tools are capable of retrieving information believed to have been deleted.

Automatic backups, storage cache, redundant storage and the cloud also help to the persistence of the information.

Two simple and clear examples of information persistence are:

- A picture uploaded into a social network remains there even after the uploaders believe they deleted them.
- Data from no longer available internet servers or storages can still be found in web search engines' caches.

B. Cryptography

According to the European Telecommunications Standards Institute (ETSI), "Without quantum-safe encryption, everything that has been transmitted, or will ever be transmitted, over a network is vulnerable to eavesdropping and public disclosure" [12].

But the privacy concerns go beyond that, once a hacker breaches the security of a system or organization, the information stored there is usually not encrypted. Wikileaks, the Panama Papers, the NSA breach and the World Anti-Doping Agency (WADA) medical records disclosure are clear examples of that.

Encrypting sensitive information within a database is not an easy or low cost task and once a hacker has gone beyond the system security, everything there is at hand and readable. We will come back this topic later when we address Format Preserving Encryption related to partial data universes in Section IX.

Discussion and comparison between symmetric and public key cryptography currently in use becomes irrelevant once one understands that none of them is unbreakable and that anything encrypted with any of them can be read if the attacker has enough computational power. Something that will happen sooner than later.

Public key algorithms such as RSA (Rivest, Shamir and Adleman), ECC (Elliptic Curve Cryptography), Diffie-Hellman and DSA (Digital Signature Algorithm) will be easily broken by quantum computers using Shor's algorithms [13] and so, they are deemed to be insecure to quantum computing.

Symmetric algorithms as AES (Advanced Encryption Standard) [14] or Blowfish [15] are believed (but not proven) to be resilient against quantum attacks by doubling the key length.

Cecilia Boschini, from IBM's Zurich Research Laboratory, was overwhelming during her presentation in IBM's annual conference Think 2018, when she emphatically affirmed that "The security our current cryptography is based, are solvable with a quantum computer".

During his talk at RSA 2018 Conference held in San Francisco, CA, USA, Konstantinos Karagiannis, CTO of Security Consulting, BT Americas, estimated that symmetric algorithms (DES, AES) with 512-bit key lengths will fall first, when the number of qubits surpasses 100.

According to Sergey Lurye from Kaspersky's Lab blog [16], "We may forecast that symmetric encryption with 512-bit keys might finally get breached by a hypothetical 144-qubit Bristlecone (Google's latest quantum processor) descendant sometime in late 2019."

Even theoretical Quantum Key Distribution (QKD) has been proved vulnerable to eavesdropping.

Any cipher that bases its strength on its complexity and in the assumption of the unavailability of the computational power required for an attack, will eventually be broken and persisting on this way will only provide a false sense of security that will last briefly.

C. Post-Quantum Cryptography

Post-Quantum Cryptography is still theoretical and far from being available.

Johannes A. Buchmann, Denis Butin, Florian Göpfert and Albrecht Petzoldt from the Technische Universität Darmstadt in their paper "Post-Quantum Cryptography: State of the Art" [17] ask and answer the question. How far is post-quantum cryptography? Their answer; "There are many promising proposals some of which are rather close to becoming practical".

Some theoretical and practical advances in Quantum Cryptography had already been proved to be vulnerable even to current non-quantum computers.

D. Vulnerability

By definition, all the cryptography in use nowadays is vulnerable to an attacker with enough computational power.

The matter is not if they can be broken but when will this happen.

Cryptography and cryptographers have been racing the Red Queen's race for a very long time. Like Alice in Lewis Carrol's "Through the Looking-Glass" [18], cryptographers have been taking all the running they can do, just to keep in the same place.

All used encryption algorithms are just temporary solutions that will eventually be rendered obsolete. A clear example of this is the Data Encryption Standard (DES) [19], it became a standard in 1977 and was broken by brute force in 1999. The Advanced Encryption Standard (AES) became a standard in 2001 and is already 17 years old.

All this because, with the only exception of the Vigenère and Vernam ciphers properly used, none of the currently used encryption solutions can answer yes to the simple question: Can this cipher resist an attacker with infinite computational power?

It is not sure whether any of the new ciphers and algorithms being developed, including those considered to be post-quantum ones, can answer yes to that same question or not.

Cryptographers will continue running the Red Queen's race as far as they continue to design complex but breakable algorithms offering only conditional and temporary security that will eventually be rendered obsolete.

E. Quantum Computingy

Quantum Computing is computing using quantum mechanics and is a field that was initiated by the work of Paul Benioff [20] and Yuri Manin [21] in 1980, Richard Feynmann [22] in 1982 and David Deutsch [23] in 1985. Current digital computers use data encoded into binary digits or bits, which can have only one value or state (0 or 1). A Quantum Bit or Qubit can have a value of 0, or 1 or 0 and 1, all at the same time.

In May 2016, International Business Machines (IBM) publicly announced they will grant access through their cloud to one of their 5 qubit quantum computers for everyone to run programs or just play with it, as a way to motivate, encourage and accelerate innovation.

In October 2017, Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, and Robert Wisnieff presented their paper "Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits" [24]. There they present calculations that were previously thought to be impossible due to impracticable memory requirements.

In November 2017, the MIT Technology Review informed and commented IBM's announcement of a 50-qubit commercial quantum computer [25].

In March 2018, Google introduced their new Bristlecone quantum processor with 72 qubits.

They are not the only ones on the field. Most governments and cutting edge technological companies and universities around the world, are dedicating time and effort in researching and investing in the development, design and manufacturing of quantum computers.

On May 2016, the European Commission announced €1 billion quantum technologies flagship project for the next ten years with the objective to reinforce European scientific leadership in quantum research and in quantum technologies.

Canadian company D-Wave [26] is already manufacturing quantum computers with two thousand qubit processors (the D-Wave 2000Q^{TM} System) and they continue improving, growing and expanding their processors.

According to CBC News, big names in the worlds of big brains and cutting edge technology like Google, NASA, Lockheed Martin and Los Alamos National Laboratory, among others, are investing big money into this company.

The Los Alamos National Laboratory's magazine 1663, on its July 2016 edition [27], published a very interesting article titled "Not Magic… Quantum", telling about a nascent commercial quantum computer that arrived to their facilities and may solve certain problems with such astonishing speed that it would be like pulling answers out of a hat.

Bjoern Lekitsch, Sebastian Weidt, Austin G. Fowler, Klaus Mølmer, Simon J. Devitt, Christof Wunderlich and Winfried K. Hensinger published the blueprint for a microwave trapped ion quantum computer in Science Advances magazine in February 2017 [28].

What we hope to achieve is to provide a technique to create ciphers offering perfect unconditional security against eavesdroppers no matter how arbitrarily powerful they may be or become in the future and without the constraints the Vigenère and Vernam ciphers have.

We want to provide a technique to create ciphers with perfect unconditional security against arbitrarily powerful eavesdroppers even if they have infinite computational power.

Something none of the currently in use standards and solutions can offer.

IV. Poly-Substitution Encryption

A. *Multiple Substitutions as Part of the Encryption*

In mono substitution encryption, the ciphertext is usually referred as the result of applying the key to the plaintext and this is not exact, which leads us to our first definition:

Definition 1. In substitution encryption, each ciphertext character is the result of applying the defined substitution and the corresponding key character's positional value to the plaintext character.

Based on that, we define the ciphertext as follows:

Definition 2. The ciphertext is the result of applying a sequence of pairs formed by the substitution and the key value to each symbol or character from the plaintext until the plaintext is exhausted.

It is crucial to understand that the ciphertext is not just the result of applying the key to the plaintext but the result of applying the sequence of pairs formed by each substitution and each key value used.

When the key is shorter than the plaintext, it wraps up at the end starts to repeat. In fact, in mono substitution encryption what starts to repeat is the same sequence of pairs formed by the substitution and the key value.

B. *Multiple substitutions*

Vigenère used modular addition as the substitution for encrypting and modular subtraction as the substitution for decrypting and those two substitutions are different, as the following example shows:

$$(17 + 23) \bmod 26 \neq (17 - 23) \bmod 26 \tag{8}$$

$$(17 + 23) \bmod 26 = 14 \tag{9}$$

$$(17 - 23) \bmod 26 = 20 \tag{10}$$

We used module 26 because Vigenère used an alphabet with 26 different symbols or characters, but an alphabet may include any number of symbols or characters with a minimum of two.

From now on, we will consider a generic alphabet called A that contains a number equal to a of different symbols or characters, being $a \geq 2$.

Formulas (5) and (6) will be generically expressed as:

$$E\,(x_n) = (x_n + k_n) \bmod a \tag{11}$$

$$D\,(x_n) = (x_n - k_n) \bmod a \tag{12}$$

Being v and v' two integer variables with values from 0 to a–1, the following formulas are always true:

$$(x_n + k_n + v) \bmod a = (x_n + k_n + v') \bmod a, \quad \text{for } v = v' \tag{13}$$

$$(x_n + k_n + v) \bmod a \neq (x_n + k_n + v') \bmod a, \quad \text{for } v \neq v' \tag{14}$$

We will represent Vigenère's encryption through the mathematical formula:

$$E\,(x_n) = (x_n + k_n + v) \bmod a \tag{15}$$

Vigenère's decryption through the mathematical formula:

$$D\,(x_n) = (x_n - k_n - v) \bmod a \tag{16}$$

So far, we have as many different encryption and decryption substitutions as symbols or characters are present in the alphabet, and as any decryption substitution can be used for encryption, we have in fact twice as many substitutions as symbols or characters in the alphabet.

If we take into consideration another mono substitution cipher as it is the Beaufort cipher, created by Sir Francis Beaufort, we can get another set of substitutions.

The Beaufort cipher, created by Sir Francis Beaufort, is a variation of the Vigenère cipher where, plaintext is subtracted from the key in order to obtain the ciphertext.

The logic is similar, only a different substitution is used, as it is shown in the following formula:

$$E\,(x_n) = (k_n - x_n) \bmod a \tag{17}$$

Applying what we have seen, such formula will become the following one:

$$E\,(x_n) = (k_n - x_n + v) \bmod a \tag{18}$$

The main difference here is that while in the Beaufort cipher the encryption and decryption substitution is the same (the plaintext is subtracted from the key to obtain the ciphertext and the ciphertext is subtracted from the key to recover the original plaintext), for $v \neq 0$, the formula for the decryption substitution will be:

$$D\,(x_n) = (k_n - x_n - v) \bmod a \tag{19}$$

For $v = 0$, the encryption and decryption substitutions are both the same.

This proves the existence of far more available encryption substitutions than symbols or characters in any given alphabet.

C. *Multiple Alphabets*

The use of more than one alphabet is possible, each with its own set of substitutions and the substitution to be used selected according to the alphabet the plaintext belongs to.

Suppose there are two alphabets, one with the letters A…Z and another one with the numbers 0…9, that way the encryption process may allow to encrypt letters into letters and numbers into numbers in a single pass.

We will come back to this later in Section IX.

D. *Fixed Substitution Sequences*

Arbitrary Sequences of substitutions can be built up with any given length, using any available substitution and placing them in any order. Each substitution may be used more than once and not all of them require to be used.

As with the key, the substitution sequence wraps up at the end when the plaintext is longer than the sequence.

If the key and the substitution sequence have the same length, then the same sequence of pairs formed by the substitution and the key value will repeat and that will make the whole encryption process vulnerable to a statistical analysis attack.

But, if the key and the substitution sequence have different lengths, when the key starts to repeat, the substitution sequence will not be the same and so there will be no sequence pair of substitution and key value at least until a position is reached within the plaintext equal to the less common multiple of the lengths of the sequence and the key.

If the less common multiple of the lengths of the substitution sequence and the key is larger than the length of the plaintext, unconditional security will be achieved, but only on such case.

E. Variable Substitution Sequences

The best and simplest way to get a variable substitution sequence is to get a fixed one and make it variable.

When hardware is constructed or software is written, the substitutions to be used are listed and located in a given order.

A list of n items can be ordered into n! (n! = 1x2x…xn) different orders and two different orders will produce two different ciphers.

Once the list is built up, the order they are listed in will not change, so, to make it variable we need an additional parameter.

There are two types of parameters we may use for that.

- External Substitution Sequence
  Instead of hardcoding the substitution sequence within the encryption process, it can be an external parameter. Doing that will allow the encryption process to use a different substitution sequence on each run. Each item on this substitution sequence will indicate which substitution will be used on each instance.
- Order Changing Parameter
  Suppose there are n different substitutions used and listed in a given order and they are numbered from 0 to n-1. That means there are n! different possible orders of the numbers from 0 to n-1. One of those permutations is loaded into an array and used as an external parameter. Each element of such array will point to a specific substitution from the list.

Using the same external substitution sequence with a different order changing parameter will produce a different substitution sequence to be used.

If every time the key and/or the substitution sequence is exhausted a new order changing parameter is used, it may be guaranteed that there will be no sequence pairs of substitution and key value repetition no matter how long the plaintext may be. We will come back to this later.

F. Variable Processing Blocks

With the substitutions we have seen so far, what any attacker will know for sure is that the first character in the ciphertext corresponds to the first character in the plaintext and so on up to the last character.

This can be avoided in a simple and elegant way. Another external parameter is used to specify a block size used to process the plaintext. As a mode of example, the block size parameter is used to define how many symbols or characters will be read at once from the plaintext and then processed in reverse order, from the last symbol or character to the first within the defined block. If the remainder of the plaintext is shorter than the last block, the block size is adjusted accordingly.

This external parameter can be a single block size or a list of different block sizes to be used along the encryption process.

Even if an attacker gets the encryption process, it provides no information about the external parameters used and so there is no way to match the plaintext symbol or character order with the ciphertext symbol or character order.

G. Poly Substitution Encryption

Now we can define what we understand for poly substitution encryption and decryption:

Definition 3. Poly Substitution Encryption is encrypting in such a way two or more different substitutions are used in sequence among the key to produce a ciphertext from the plaintext.

Definition 4. Poly Substitution Decryption is decrypting in such a way two or more different substitutions are used in sequence among the key to retrieve the original plaintext form the ciphertext.

V. The Seven Steps

We defined the process to build up ciphers based on our technique as a step by step process comprised of 7 steps.

A. Step One (Use Multiple Encryption Substitutions)

While Vigenère used modular addition module 26 as the substitution and Vernam used a single function (XOR) as the substitution. Our approach will use many of them. Each substitution will take a plaintext character and a key character and will return a ciphertext character and for each of the possible key character values will return a different ciphertext character.

Using multiple substitutions provides additional security because, if an attacker has a cryptogram or ciphertext character or symbol, not only the key character used is unknown, but also the substitution used.

For a given plaintext byte character, any valid substitution should return different results for each possible value of the key character.

When the plaintext's length is larger than one character we can use one substitution to process the first byte, the same or a different one to process the second character and so on. That leads us to the following step.

B. Step Two (Use a Third Parameter)

The first two parameters will be the plaintext and the key.

A third parameter will be used to indicate which substitution to use on each instance.

A value from this third parameter will indicate which one of the many available substitutions will be used to process a character or symbol from the plaintext and one from the key.

Let us say we decide to use the same number of different substitutions from all that can be created as the number of symbols or characters in the used alphabet. In such case, we will only need one character from this third parameter to indicate which of those substitutions will be used for these specific plaintext and key characters. So far, the third parameter character value x will trigger substitution z. How do we know which of the available substitutions is substitution z, is explained in the next step.

C. Step Three (Order of the Substitutions)

When we have many different substitutions, we need to identify them somehow and make a list of them.

This list is what will be used to decide which substitution will be triggered by which value from the third parameter.

As previously indicated, this list is not unique and a different substitution order will produce a different ciphertext for the same plaintext and key.

Now, an attacker not only needs to try every possible key, also needs to try every possible third parameter and guess which substitution is triggered by each possible value of the third parameter, assuming the selected substitution order is hardcoded within the process.

So far, parameter byte value x will always trigger substitution z, unless we can make parameter value x trigger substitution w in a different run.

The order of the substitutions can be changed, as explained in the next step.

D. Step Four (Changing the Order of the Substitutions)

How do we make third parameter byte value x to trigger a substitution different from substitution z?

The solution is both simple and elegant.

We add a fourth parameter. One of those n! possible orders of the numbers from 0 to n-1 is loaded into a n element array, and value x is used to point to the array's element whose value will be used to trigger the substitution.

A different fourth parameter will provide a different substitution order.

Now, third parameter character value x will trigger a substitution depending on the x^th^ element of the fourth parameter.

So far, any attacker would know that the first byte from the ciphertext corresponds to the first byte of the plaintext, the second byte from the ciphertext corresponds to the second byte of the plaintext, and so on.

The next step will show how to change that.

E. Step Five (Block Processing)

Let us take a block of characters of a given length from the plaintext and process it in reverse order, starting from the last symbol or character in the block, processing it and saving it as the first character in the ciphertext block. Then the previous to the last to be the second character in the ciphertext block and so on, until we end processing the block by processing its first character and then continue with the next block.

The last block may be shorter but it is equally processed from last character to first one as any other block without any need of any padding or additional dummy information to be added.

Now, unless the attacker knows the exact length of the block used, there is no way to know from where to start to retrieve from the ciphertext to obtain the original plaintext in the original order.

F. Step Six (Key Length and Key Repetitions)

So far, no mention has been made of the key length.

When encrypting, the process uses two items with the plaintext: the key and the encryption substitution. So, for each portion of the plaintext, a key-substitution pair is used. This is usually ignored due to the encryption substitution being always the same.

Vigenère's and Vernam's ciphers require the key to have at least the same length as the plaintext for them to offer unconditional security. If the key is shorter, the process starts to repeat the same key-substitution pair sequence and this weakens its security and makes a statistical distribution analysis attack feasible.

If we use a key shorter than the plaintext it will wrap up at the end, but unless the key and the third parameter both have the exact same length, there will be no same key-substitution pair sequence repetitions until we reach a position within the plaintext equal to the least common multiple of the lengths of both the key and the third parameter. As it may eventually happen the whole process would be vulnerable unless we find a way to avoid repetitions.

The solution is, once again, simple and elegant. When the end of the key is reached (or the end of the third parameter or the less common multiple of both lengths, or at any point between them), before starting to repeat it, the process changes the substitution order by modifying the elements in the array explained in step four.

One way to do it is to use the last used substitution and last used plaintext or ciphertext symbol or character and apply that transformation to each element of the array using the element's content and the plaintext character as input and replacing the content of the element with the result, thus obtaining a different permutation of the array elements.

Each time this happens, the change process behaves differently and a different permutation is obtained. Now, even if the key and the third parameter have the exact same length and they start to repeat in the exact same order, the sequence of key-substitution pairs triggered will not be the same and so no repetitions will occur. The same third parameter value will point to the same array element but a different substitution will then be triggered because the content of the array element will have changed and so the second parameter-substitution pair sequence will be totally different.

G. Step Seven (Make Lengths Variable)

Current encryption standards use fixed length blocks and fixed length keys (they may offer different key sizes but with very limited pre-defined fixed sizes).

Our solution allows for user selected lengths for the key, the third parameter and the processing block (or blocks). Two successive encryptions may use not only different keys but also the lengths of both keys may be different. The same applies to the third parameter and also the processing block size may be different. The key length may go from a single character to any length, even the same length of the plaintext or longer. The third parameter may go from a single character to any length, even the same length of the plaintext or longer. The processing block size may go from a single character to any length up to the length of the plaintext and is limited only by the maximum size allowed by the system where the encryption is implemented. It is also possible to process consecutive blocks of different sizes by using a sequence of values instead of a fixed one, indicating the individual size for each individual block to be processed. When the last processing size list element is exhausted, it wraps up and starts over from the beginning. When building up an application, different groups and number of substitutions may be used to create personalized non-standard versions.

VI. Building Up A Poly Substitution Cipher

A. A cipher complying with these seven steps

In order to be able to make comparisons with known standards, we decided to use a standard alphabet of 26 letters (A…Z).

We built up a cipher accordingly and complying with these seven steps and it uses five parameters:

- The plaintext to encrypt
  The plaintext is just a sequence of characters of any length.
- The key to be used.
  This key is just a sequence of characters of any length and can be longer, equal in length or shorter than the plaintext.
- A third parameter defining which substitution to use on each instance.
  This third parameter is a sequence of characters of any length and there is no required relation between its length and the lengths of the plaintext or the key.
- An initial substitution order.
  This is a sequence of values that will be used to define an initial order for the encryption substitutions to be used.
- A processing block size.
  This will define the number of characters to be read at once from the plaintext and processed in reverse order (from the last character to the first one) to generate the ciphertext. A value of 1 (one) will make the plaintext to be processed straight from the first character to the last one.
  This can be a fixed value to process same size blocks (all but maybe the last one) or a sequence of values to indicate the size of each individual block to be processed.

Depending on how the cipher is programmed and implemented, it can allow the user to manually type every parameter or to select or chose them.

The encryption process will work as follows:

1. The user may select or enter the plaintext to process, the key, the third parameter, the initial substitution order and the processing block size or sizes.
2. The process loads the initial substitution order into an array with the same number of elements as substitutions to be used.
3. If the remaining of the plaintext is shorter than the processing block, the processing block size is adjusted accordingly.
4. The process reads a processing block from the plaintext. If the plaintext has been exhausted, the process ends.
5. The process takes the last character from the processing block.
6. The process takes a character from the key.
   If the key has been exhausted, reorder the initial substitution order array elements and read the first key byte again.
7. The process takes a character from the third parameter.
   If the third parameter has been exhausted, start over from its first character.
8. The process uses the character from the third parameter to point to an element from the substitution order array and uses its value to trigger an encryption substitution passing the plaintext and key characters as parameters.
9. The substitution triggered returns a ciphertext character that is written to the ciphertext output.
10. The process takes the previous character from the processing block.
    If the processing block has been exhausted, jump to step 3.
11. Jump to step 5.

The decryption process will work the exact same way, using the ciphertext instead of the plaintext and reversing the encryption process, using the same key, and same remaining parameters, using the reverse substitution on each instance.

B. Typical Distribution of Letters in English Language

There is not an unique and generic distribution of letters in the English language and the use of field-related jargons may impact the results. (i.e., a statistical analysis of medical books may provide a different result from financial or sports books). Despite those small differences, the results are mostly similar on which are the most common letters. Figure 5 shows the relative frequencies of letters in the English language based on “Letter Frequencies in English”, published by The Oxford Math Center [29] from Oxford College of Emory University.

Letter	Frequency (in %)
A	8.167%
B	1.492%
C	2.782%
D	4.253%
E	12.702%
F	2.228%
G	2.015%
H	6.094%
I	6.966%
J	0.153%
K	0.772%
L	4.025%
M	2.406%
N	6.749%
O	7.507%
P	1.929%
Q	0.095%
R	5.987%
S	6.327%
T	9.056%
U	2.758%
V	0.978%
W	2.360%
X	0.150%
Y	1.974%
Z	0.074%

Figure 5. Relative Frequencies of Letters in English.

These letter frequency values can also be graphically represented as it is shown in Figure 6, which is very similar to the one published by Wikipedia [30].

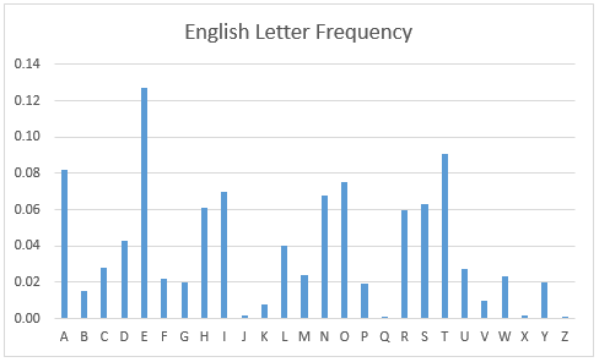

Figure 6. English Letters Frequency Graph.

C. *Our Test*

We took a text file from Project Gutemberg containing the Complete Works from Winston Churchill (the American Winston Churchill, not the British one) [31]. A simple text file 9,540,229 characters long from which we removed all non-alphabetic characters and obtained a 7,221,951 character long alphabetic only text.

We selected the 13 character long non-random key “WHENIWASYOUNG” and the 15 character long non-random value “WEONLYJUSTBEGUN” as the third parameter and used a fixed block size of 1 byte to keep the character sequence between the plaintext and the ciphertext.

From all 26 that form the English alphabet, the key uses only 11 different letters (A, E, G, H, I, N, O, S, U, W and Y) while the third parameters uses only 12 different letters (B, E, G, J, L, N, O, S, T, U, W, and Y). Both parameters share 8 letters in common (E, G, N, O, S, U, W and Y).

The less common multiple of the lengths of the key and the third parameter is 195. The same sequence pair of third parameter character and key character repeats over 37,035 times to match the plaintext file length.

We performed a frequency distribution analysis of the source file and the results are listed in Figure 7.

Letter	Frequency (in %)	Frequency
A	8.0788%	583,448
B	1.4146%	102,161
C	2.4972%	180,350
D	4.5871%	331,279
E	12.6098%	910,672
F	2.1128%	152,585
G	2.0429%	147,538
H	6.6493%	480,210
I	6.9210%	499,830
J	0.1486%	10,732
K	0.8086%	58,399
L	3.8873%	280,738
M	2.6948%	194,620
N	6.9766%	503,850
O	7.5462%	544,985
P	1.6051%	115,918
Q	0.0815%	5,884
R	5.8154%	419,982
S	6.1690%	445,519
T	9.0004%	650,006
U	2.7707%	200,100
V	0.9488%	68,521
W	2.4342%	175,795
X	0.1404%	10,140
Y	1.9878%	143,558
Z	0.0710%	5,131

Figure 7. Plaintext File English Letter Frequency in numbers.

The result is not an exact match but the differences are minimal and the order of the four most common letters (E, T, A and O), is the same. Based on those results, we generated a graphical representation, which is presented in Figure 8 and matches the standard graphical distribution from Figure 6.

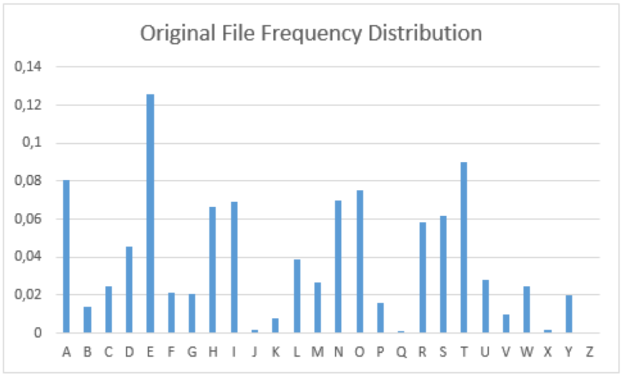

Figure 8. Plaintext File Statistical Letter Frequency Graph.

There are 7,221,950 bigrams formed by two consecutive letters in the plaintext and 7,221,949 trigrams formed by three consecutive letters. Figure 9 shows the results for the 30 most common bigrams found in the source file (those that are repeated more times along the plaintext).

Order	Bigram	Frequency in %	Frequency
1	TH	3.0676%	221,539
2	HE	3.0159%	217,807
3	IN	1.7960%	129,709
4	ER	1.7902%	129,286
5	AN	1.6384%	118,326
6	RE	1.3856%	100,070
7	ED	1.3314%	96,154
8	HA	1.2584%	90,882
9	ES	1.2079%	87,233
10	TO	1.2042%	86,970
11	ND	1.1985%	86,552
12	EN	1.1590%	83,704
13	OU	1.1378%	82,169
14	NT	1.1341%	81,903
15	ON	1.1195%	80,850
16	AT	1.0931%	78,943
17	ST	1.0649%	76,909
18	EA	0.9474%	68,424
19	HI	0.9456%	68,293
20	IT	0.9145%	66,047
21	AS	0.8947%	64,612
22	ET	0.8520%	61,530
23	OR	0.8452%	61,039
24	NG	0.8423%	60,833
25	IS	0.8241%	59,517
26	TE	0.7945%	57,378
27	AR	0.7714%	55,710
28	TI	0.7699%	55,600
29	OF	0.7433%	53,678
30	SE	0.7316%	52,837

Figure 9. Source File Bigram Frequency.

On the other extreme of the listing, from all 676 possible bigrams, there are 97 that are repeated 10 times or less within the plaintext. From them, 49 of them are not present at all, 11 are present only once, 9 appear twice, 4 appear three times, 4 appear four times and 4 appear five times. The results from the plaintext will be compared with the results from the ciphertext.

Figure 10 shows the results for the 30 most common trigrams found in the source file.

Order	Trigram	Frequency in %	Frequency
1	THE	1.8690%	134,980
2	AND	0.8596%	62,080
3	ING	0.6603%	47,687
4	HER	0.6143%	44,362
5	THA	0.4724%	34,118
6	ERE	0.4350%	31,419
7	HAT	0.4128%	29,811
8	YOU	0.3814%	27,542
9	NTH	0.3670%	26,504
10	ENT	0.3650%	26,363
11	WAS	0.3440%	24,845
12	SHE	0.3425%	24,733
13	HIS	0.3362%	24,278
14	ETH	0.3251%	23,478
15	HES	0.3168%	22,876
16	DTH	0.3046%	21,999
17	THI	0.2988%	21,577
18	INT	0.2934%	21,190
19	FOR	0.2912%	21,030
20	ITH	0.2707%	19,548
21	HAD	0.2693%	19,450
22	TTH	0.2476%	17,880
23	TER	0.2392%	17,272
24	ION	0.2372%	17,128
25	OFT	0.2360%	17,047
26	FTH	0.2351%	16,981
27	EST	0.2327%	16,807
28	OTH	0.2315%	16,716
29	EDT	0.2286%	16,507
30	WIT	0.2232%	16,119

Figure 10. Source File Trigram Frequency.

On the other extreme of the listing, from all 17,576 possible trigrams, 8642 are not present at all, meaning that almost half of all possible trigrams (49.17%) are not present in the plaintext.

After encrypting the file, we performed the same frequency distribution analysis on the encrypted file as we did on the original source text file. Figure 11 shows the encrypted file letter frequencies graphical distribution (showing frequency distribution as a percentage of the total number of characters).

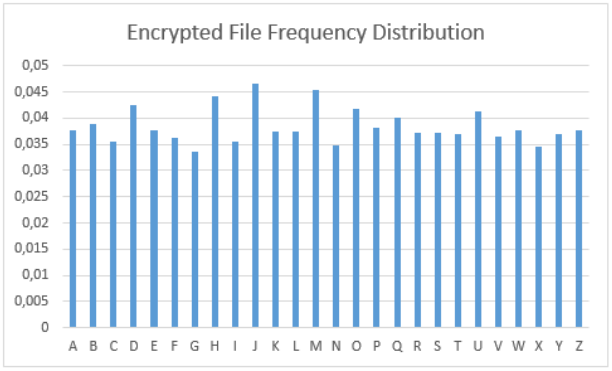

Figure 11. Ciphertext File Statistical Letter Frequency Graph.

The letter frequency of the encrypted file is homogeneous and almost flat making impossible a statistical distribution analysis attack based on the statistical distribution of the letters in the English language. There is no way to match the letters from the encrypted file with those from the source file. Figure 12 shows the letter frequency from the ciphertext.

Letter	Frequency (in %)	Frequency
A	3.7686%	272,167
B	3.8798%	280,198
C	3.5510%	256,452
D	4.2553%	307,314
E	3.7649%	271,896
F	3.6380%	262,733
G	3.3720%	243,523
H	4.4154%	318,879
I	3.5593%	257,049
J	4.6710%	337,340
K	3.7397%	270,079
L	3.7402%	270,112
M	4.5380%	327,730
N	3.4808%	251,384
O	4.1863%	302,336
P	3.8112%	275,244
Q	4.0089%	289,520
R	3.7138%	268,209
S	3.7167%	268,416
T	3.7002%	267,229
U	4.1338%	298,538
V	3.6556%	264,003
W	3.7597%	271,523
X	3.4586%	249,776
Y	3.7073%	267,740
Z	3.7741%	272,561

Figure 12. Ciphertext File Letter Frequency.

When we order the letters from the plaintext and the ciphertext sorting them down in decreasing order of the statistical distribution of the letters within them, we get the graphical representation displayed in Figure 13.

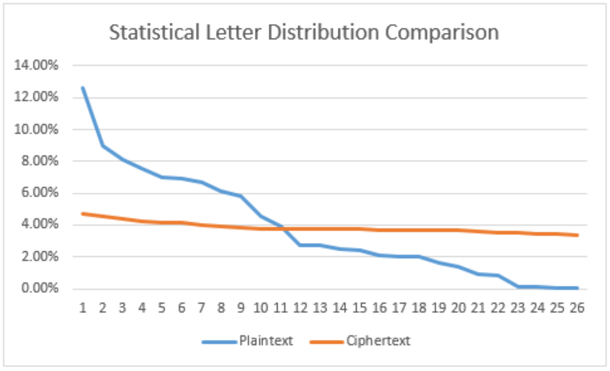

Figure 13. Statistical Letter Distribution Comparison.

We analyzed the frequency distribution of bigrams and trigrams within our encrypted text file and Figure 14 shows the results for the 30 most common bigrams in the ciphertext.

Order	Bigram	Frequency in %	Frequency
1	JJ	0.2181%	15,750
2	JM	0.2118%	15,299
3	MJ	0.2096%	15,136
4	JH	0.2055%	14,843
5	HJ	0.2040%	14,734
6	MM	0.2022%	14,603
7	MH	0.2021%	14,596
8	JD	0.2007%	14,492
9	JO	0.1997%	14,422
10	HM	0.1995%	14,411
11	MD	0.1968%	14,211
12	DJ	0.1962%	14,168
13	OJ	0.1957%	14,130
14	HH	0.1955%	14,116
15	JU	0.1944%	14,036
16	UJ	0.1940%	14,008
17	DM	0.1938%	13,999
18	MU	0.1904%	13,753
19	QJ	0.1897%	13,701
20	JQ	0.1888%	13,632
21	MO	0.1883%	13,602
22	OM	0.1878%	13,565
23	UM	0.1868%	13,494
24	DH	0.1864%	13,461
25	OH	0.1857%	13,412
26	HU	0.1853%	13,384
27	HO	0.1851%	13,369
28	QM	0.1843%	13,310
29	HD	0.1836%	13,257
30	MQ	0.1826%	13,185

Figure 14. Encrypted File Bigram Frequency.

The first finding analyzing bigrams was that all possible 676 bigrams are present in the ciphertext, the most repeated one appears 15,750 times and the least repeated one appears

8,223 times in it. The bigram distribution for the encrypted file is almost flat and homogeneous and there is no way to match the bigrams from the encrypted file with those from the source file as Figure 15 shows.

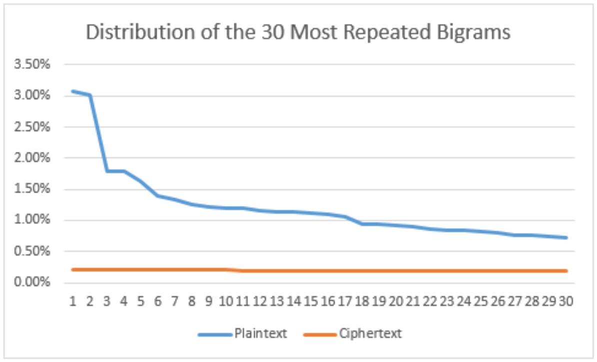

Figure 15. Bigram Statistical Frequency Comparison.

Figure 16 shows the results for the 30 most common trigrams found in the encrypted file.

Order	Trigram	Frequency in %	Frequency
1	MMJ	0.0106%	763
2	HJJ	0.0105%	759
3	MJJ	0.0105%	755
4	JJM	0.0102%	739
5	JMJ	0.0101%	733
6	MJH	0.0101%	727
7	JJH	0.0101%	726
8	MDM	0.0100%	722
9	JJJ	0.0099%	718
10	JJD	0.0098%	710
11	MHJ	0.0098%	710
12	HJM	0.0098%	709
13	JHM	0.0098%	707
14	MMM	0.0097%	704
15	JMM	0.0097%	701
16	JMD	0.0097%	698
17	JDM	0.0097%	697
18	MHH	0.0096%	695
19	JHH	0.0096%	694
20	JJU	0.0096%	691
21	MJO	0.0096%	691
22	JEJ	0.0095%	688
23	HMJ	0.0095%	687
24	UJO	0.0095%	687
25	JOJ	0.0095%	684
26	JOH	0.0095%	683
27	DJJ	0.0094%	681
28	MPJ	0.0094%	681
29	YJJ	0.0093%	675
30	HHJ	0.0093%	673

Figure 16. Encrypted File Trigram Frequency.

The first finding analyzing trigrams was that all possible 17,576 trigrams are present in the ciphertext, the most repeated once appears 763 times and 260 times the least repeated one. The most repeated bigram (MMJ) repeats less than three times the least repeated one (GGV) and the flatness of the distribution makes unfeasible any statistical analysis attack to retrieve the original plaintext from the ciphertext based on the distribution of trigrams.

Comparing the statistical distribution of the 30 most common trigrams from the plaintext and the ciphertext, Figure 17 shows the differences that make unfeasible a statistical analysis attack based on the distribution of trigrams.

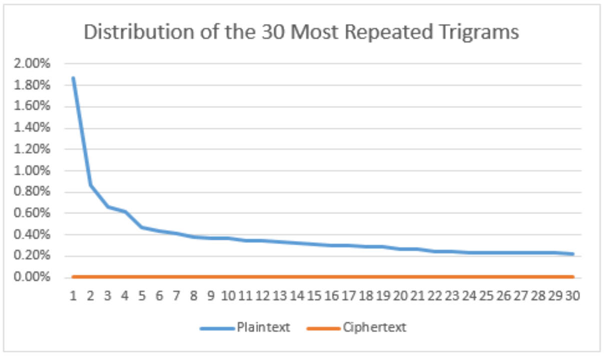

Figure 17. Trigram Statistical Frequency Comparison.

Going one step further and considering tetragrams (sets of four consecutive letters), there are 456,976 possible tetragrams and from them 378,431 are not present in the plaintext. Only 17.188% from all possible tetragrams appear in the plaintext being THAT with a 0.3171% (22,898 repetitions) and THER with 0.3089% (22,305 repetitions) the two most common of them. From all 456,976 possible tetragrams, only two are not present within the ciphertext (HXYA and NGXC), being HJMJ with a 0.00071% (51 repetitions) and HHJJ with 0.00068% (49 repetitions) the two most common of them.

Figure 18 compares the statistical distribution of the 26 most common tetragrams from the plaintext and the ciphertext.

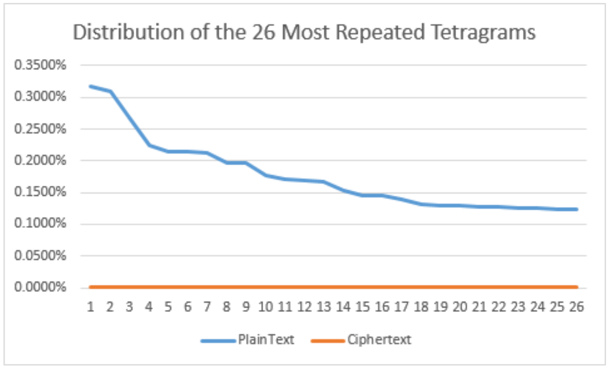

Figure 18. Tetragram Statistical Frequency Comparison.

D. Results of Our Test

Although the key used was very short (only 13 characters long) and not random, the third parameter was also very short (only 15 characters long) and also not random, and the source file was pretty big (over 7 Megabytes long), the use of poly substitution encryption provided a level of confusion and diffusion that makes any cryptanalysis or statistical distribution analysis attack totally unfeasible, as will be proved in Section VIII.

When the gross frequency (the number of times each item is repeated as a number, not as a percentage) of letters, bigrams, trigrams and tetragrams are compared between the plaintext and the ciphertext, the flatness of the results gets visually clear.

In the distribution of letters within the plaintext (alphabetically ordered from A to Z), as shown in Figure 19, the most common letters E, T, A and O are clearly distinguishable.

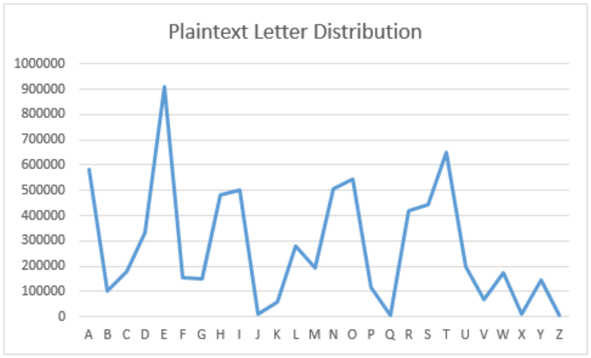

Figure 19. Plaintext Letter Distribution.

A ciphertext obtained from a Vigenère encryption of the plaintext using the same key, would have produced the exact same graph. The graphic of the letter distribution within the ciphertext displayed in Figure 20, shows not only the flatness of the distribution obtained through the use of a cipher based on our proposed encryption, but the small variation among the letters.

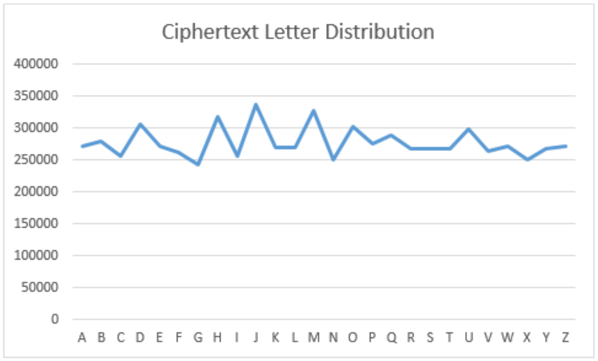

Figure 20. Ciphertext Letter Distribution.

In the distribution of bigrams within the plaintext (ordered from AA to ZZ), as shown in Figure 21, the most common bigrams TH and HE are clearly distinguishable.

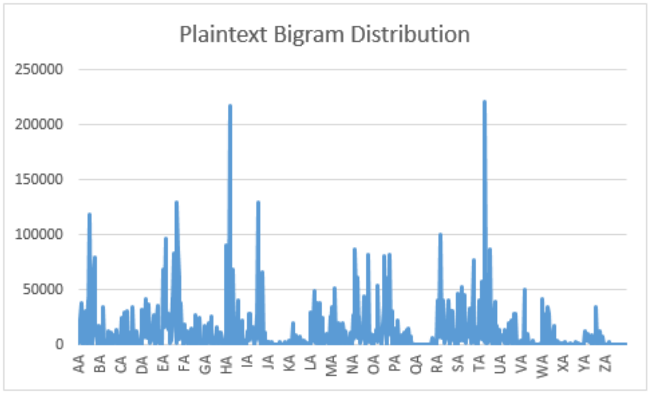

Figure 21. Plaintext Bigram Distribution.

The distribution of bigrams within the ciphertext also shows the flatness of the result and the low variation between the different bigrams, as shown in Figure 22.

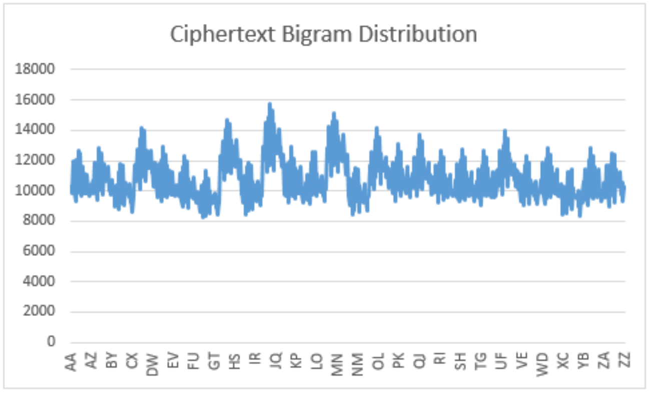

Figure 22. Ciphertext Bigram Distribution.

Although some bigrams are not present in the plaintext, all possible bigrams are present in the ciphertext.

In the distribution of trigrams within the plaintext (from AAA to ZZZ), as it is shown in Figure 23, the most common values THE, AND, ING and HER, are clearly distinguishable.

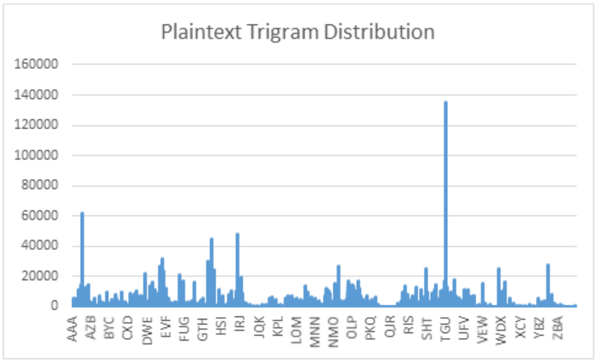

Figure 23. Plaintext Trigram Distribution.

But the distribution of trigrams within the ciphertext (from AAA to ZZZ), as it is shown in Figure 24, tells a totally different story. Once again, the graphic of the trigram distribution within the ciphertext displayed in Figure 24, shows not only the flatness of the distribution obtained through the use of a cipher based on our proposed encryption, but the small variation among the different trigrams.

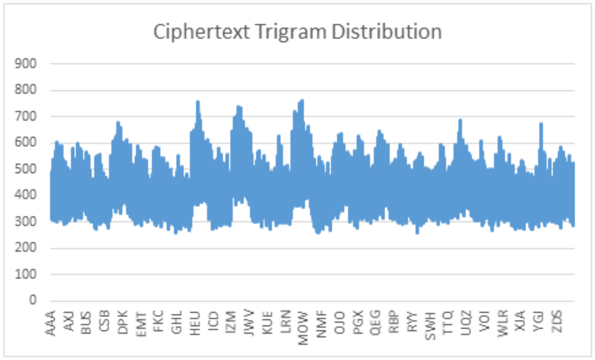

Figure 24. Ciphertext Trigram Distribution.

The graphical representation of the ciphertext trigram distribution resembles white noise. White noise is basically pure random noise that has all the frequencies in the audio spectrum. It is a random signal having equal intensity at different frequencies, giving it a constant power spectral density. Different sounds and musical notes are produced by a combination of different frequencies, the same way as words and phrases are formed by a combination of different letters from an alphabet. Some random number generators are based on white noise like the one used by Random.org, whom uses a system of atmospheric antennae to generate random digit patterns from white noise. White noise sounds pretty close to static from your old television set or radio when no station was tuned or a whooshing sound. Additive white Gaussian noise (AWGN) is a basic noise model used in information theory (originally proposed by Claude E. Shannon in 1948 in his landmark paper [7]) to mimic the effect of many random processes that occur in nature.

This shows how the distribution of trigrams within the ciphertext cannot be distinguished from being totally random. Considering that the same occurs with the distribution of bigrams and single letters within the ciphertext, that proves that any statistical analysis attack on the ciphertext will not succeed as we will prove in Section VIII.

VII. Analysis

A. *Comparing this cipher with Vigenère's and Vernam's*

It is trivial to prove that the Vigenère cipher is a limited or restricted version of our proposed encryption where the processing block size is one character and only one substitution is used (modular addition with v = 0), which means the third and fourth parameters need to be built in a way that ensure the same substitution will always be triggered.

It has already been proved that the Vernam cipher is a restricted or limited version of the Vigenère cipher, where the alphabet used has only two characters or symbols).

A text message properly ciphered through the Vigenère or Vernam Ciphers (using a random one-time key as long as the plaintext) gives absolutely no clue on the key used or the original plaintext and a brute force attack will end up with a huge number of false positives.

A brute force attack will return some invalid or unreadable results but will also return any possible message with the exact same length and there is no way to decide which one is the true original one.

Vigenère and Vernam ciphers are not used because they have the same three requirements that need to be fulfilled to comply with Shannon's definition for Perfect Secrecy:

1) The key needs to have the same length as the plaintext.
2) The key must be random.
3) The key must not be reused.

These three requirements are mandatory because Vigenère and Vernam used a single encryption substitution (Modular Addition and XOR) in the process.

With the Vernam cipher, for any given ciphertext byte, the attacker needs to try every possible key byte value and will end up with 256 different results, each one with the exact same probability of being the plaintext byte value.

With the Vigenère cipher, for any given ciphertext symbol, the attacker needs to try every possible key symbol and will end up with every possible symbol in the alphabet, each one with the exact same probability of being the plaintext symbol.

With our proposed encryption technique and even assuming the attacker knows the exact processing block size used for this specific ciphe text, all the encryption substitutions used and can match each ciphertext byte with the corresponding byte position in the plaintext; the attacker will still need to try each of the 256 possible key byte values with each of the encryption substitutions involved. So, if we used 256 different encryption substitutions, the attacker will end up with 65,536 possible values for the plaintext byte, each one repeated many times and no way to decide which value is the original one.

If the attacker does not know the processing block size, it multiplies the effort required as the first byte from the plaintext may correspond to any of the bytes in the ciphertext, the second one to any of the bytes except the last and so on, doing the math it means there are n! possible orders for the ciphertext to match the byte order of the plaintext, being n the length in bytes of both the plaintext and the ciphertext.

Our proposed cipher does not have any constraints as Vigenère and Vernam ones do, and we will prove that our encryption overcomes those constraints that come from mono-substitution encryption limitations.

As we can assure the same key value-encryption substitution sequence will not be repeated, the length of the key becomes irrelevant, it may have any length and it does not matter if it is shorter than the plaintext.

So far, we have been able to overcome the first of Vigenère's and Vernam's constraints and now the key can be shorter than the plaintext without impacting the safety of the process.

As we use some additional parameters, does the key truly need to be random?

Leaving aside any discussion about what is truly random and what is not, anything can be used as a key; a text, a web page, a file from the Internet. As far as the key is kept secret, it really does not matter whether it is truly random or not.

In the example we provided the key used was extremely short compared to the length of the plaintext (the plaintext length was about half a million times the length of the key) and it was not random at all and despite that, no statistical analysis attack will succeed in retrieving the original plaintext having only the ciphertext.

As the process does not use an unique encryption substitution but a pool of them, the randomness of the key has no impact on the outcome because the substitution sequence order cannot be predicted.

Our test showed how using a short non-random key had no impact on the security of the process.

So far, we have been able to overcome the second of Vigenère's and Vernam's constraints and now the key does not need to be random.

What if the key is reused?

As we consider the encryption process as the result from applying a sequence of key value-encryption substitution pairs, what we need to avoid is reusing that exact same sequence.

It is clear that if any of the third parameter, the initial substitution order or the block size (or sizes) is different, the key value-encryption substitution pair sequence will be different for the same key. If the same substitution order array rearrangement used when the key is exhausted is used every time a processing block is exhausted, it can be guaranteed that the same key value-encryption substitution pair sequence will not be repeated for a different plaintext even if the exact same parameters are used and if the first processing block is just one byte long, then no other key value-encryption substitution pair sequence will be repeated beyond the first symbol even if the same exact parameters are reused. This happens because the probability that two different plaintexts or ciphertexts may have matching characters in the exact same position every time the key, the processing block or whatever trigger may be used is exhausted, is just zero.

This overcomes the third of Vigenère's and Vernam's constraints and now the key may be reused without compromising the unconditional security.

As you see, an encryption algorithm based on our proposed technique and complying with its seven steps can guarantee unconditional security using non-random keys that can be shorter than the plaintext and can be reused.

Also, two successive encryptions may exchange the key with the third parameter using the third parameter as the new key and the old key as a new third parameter keeping the unconditional security.

As far as the key does not need to be random, selecting a new different key is quite easy. There is no need of a random or pseudo-random key generator as any possible file may be used as a key (or third parameter). A web page from any site, a file of any type or even portions of them can be used.

Been Vernam the only cipher mathematically proved to be unbreakable if properly used, let us do a comparison between a cipher based on our proposed poly-substitution encryption and the Vernam cipher. Figure 25 shows a comparison between Vernam's cipher and our proposed one.

	VERNAM	Cipher based on our Proposed Technique
Sample plaintext length	140	140
Processing block size	1	Variable
Key size	140	Variable
Key and plaintext length must match	Yes	No
Key must be true random	Yes	No
Key must not be reused	Yes	No
No. of substitutions	1	256
No. of possible results per substitution	256	256
No. of possible results per Byte	256	65,536
Ciphertext to plaintext match	1	140!
No. of possible results per Byte from brute force attack	256	65,536 x 140!
No. of possible results per Byte from brute force attack as power of 2	2^(8)	2^(809)
Probability of being the plaintext byte	0,39%	0,39%
Rounds	1	1
False Positives	Yes	Yes

Figure 25. Comparing Vernam's cipher to our proposal.

An encryption algorithm based on our technique and complying with the seven steps will offer the same unconditional security offered by the Vernam cipher without the constraints it has.

A plaintext encrypted with such algorithm will remain impervious to any attack, no matter how powerful the attacker may be or may become, even if the attacker has infinite computational power.

B. Comparing this cipher with currently used ciphers

Due to their extreme complexity, none of the current encryption standards will produce a false positive when an incorrect or wrong key is used.

All currently used encryption base their privacy and security on the unavailability of enough computational power required to try all possible keys in a short time and that is why they will all fall under a quantum attack capable of trying every possible key in very little time.

There is an old saying: "How do you hide an elephant on a beach? By filling the beach with elephants".

The strength of our proposed encryption technique relies not on the computational power required to try every possible key, third parameter, initial substitution order, substitution set or block size or sizes; its strength relies on the fact that we assume it can be done but the real original plaintext will be hidden at plain sight within an immense sea of false positives with absolutely no indication on which one is the right one. It is not that the attacker will not be able to get the original plaintext, it is that the attacker will not be able to distinguish the original plaintext from the false positives.

We assume the attacker will reach the beach and see all that is there. There can also be palms and monkeys, and seagulls and turtles, and crabs and people and of course sand and water on that beach, but even if the attacker knows that what is hidden there is only an elephant, still will have no clue and no way to know which one of all the elephants there is the right one. Even if the attacker is capable of knowing that what is hidden there is an elephant and ignores all the palms and monkeys and seagulls and turtles and crabs and people and any other element that is not what is hidden there, even then, there is absolutely no way and no clue to know which elephant is the right one.

Symmetric key cryptography relies on very complex processes where a brute force attack may need too much computational power to succeed and no such power is yet available. They do not say their ciphers cannot be broken with enough computational power, they claim and hope for the required computational power not to be available anytime soon.

Besides the existence of some known partial attacks that may succeed with more computational power like quantum computers promise to deliver, the use of a fixed size processing block and fixed size keys (they may offer different key sizes but limited to very few pre-defined options. They do not allow the user to freely choose any key size, less to select the block size), make them vulnerable to brute force attacks given the attacker has enough computational power.

While currently in use symmetric encryption standards use block sizes of 64 or 128 bits and keys of 128 to 512 bits, our encryption handles blocks of any size including successive blocks of different sizes and keys of any length. Figure 26 shows a comparison between a cipher based on our proposed technique and other symmetric ciphers.

	Block Size	Key Size	Rounds	False Positives
DES	64 bit	56 bit	16	No
3DES [33]	64 bit	128 bit	48	No
AES	128 bit	128, 192, 256 bit	9, 11, 13	No
BLOWFISH [34]	64 bit	32-448 bit	16	No
Cipher based on our Proposed Technique	Variable	Variable	1	Yes

Figure 26. Comparison between a cipher based on this technique and current symmetric standards.

Public-key cryptography currently in use (including RSA and ECC) relies on the assumption that some problems cannot be solved or would will require an extremely long time to be solved, and therefore, that it would take a very long time for their secured data to be decrypted. But as quantum algorithms can solve some of these problems with ease, that assumption is fatally challenged. It is known that a quantum computer running Shor's algorithms can easily solve complex problems like long integer factorization and discrete logarithms, which are the foundation of public key cryptography.

VIII. Attacking a Cipher Based on This Technique

Trying to retrieve the plaintext from a ciphertext created through an implementation of this technique without having any additional information will be at least as difficult as trying to retrieve the plaintext from a one-time pad created ciphertext or one created through a proper use of Vigenère's or Vernam's ciphers having only the ciphertext.

Any attack must take into consideration that all the parameters are external to the process and they all may be different from one encryption to the next and also the fact that the process may be used in reverse order. Decryption can be used to protect the plaintext and encryption with the same parameters used to retrieve the original plaintext.

As cryptanalysis of our encryption is just not possible, any possible attacker will need to face the following difficulties when attempting a brute force attack to break an encryption created with a cipher based on this technique and complying with its seven steps:

- Which ciphertext byte corresponds to each plaintext byte.
- Which encryption substitutions exist and which of them were used.
- Which substitution was used on each instance.
- Which was the key used.
- Which processing block size or sizes were used.

Let us give the attacker the advantage of knowing all the encryption substitutions involved, the specific set used to create the ciphertext and the processing block size or sizes used. In such situation, for each byte in the ciphertext, the attacker needs to try every possible substitution for every possible key byte value and so, instead of getting 256 possible values as with Vigenère's or Vernam's ciphers, the result will be 65,536 possible values having every single one the exact same probability of being the plaintext byte value despite the repetitions.

That is the best case scenario for the attacker.

If the processing block size or the sequence of block sizes is not known, the attacker will need to try any possible fixed or variable block size from a single byte to the length of the ciphertext. While this adds time and difficulty to the attack, every possible outcome still has the exact same probability of being the original plaintext despite the repetitions.

The original plaintext will still be lost in a sea of false positives with no way to decide which one is the right one. The beach will remain full of elephants with no clue on which one is the right one.

As Ronald Linn Rivest, Adi Shamir and Leonard Adleman stated on their seminal paper [35], "Since no techniques exist to prove that an encryption scheme is secure, the only test available is to see whether anyone can think of a way to break it." Based on that, we will give a try to classical and modern cryptanalysis attacks and try to show how and why they will not succeed in breaking the encryption.

A. Statistical Letter Distribution Attack

We have seen that single character distribution within the ciphertext is practically flat, making impossible to identify the original corresponding characters from the plaintext through a statistical distribution attack.

B. Kasiski Statistical Analysis Attack

A Kasiski Statistical Analysis attack uses repeated sequences from the ciphertext to try to narrow down the length of the key.

First we analyzed the 10 most common bigrams (JJ, JM, MJ, JH, HJ, MM, JD, JO and HM) and measured the distances between consecutive occurrences of the same bigram across the ciphertext and found out that for all of them, the minimum distance between two occurrences of the same bigram is zero, meaning that each bigram shows two times consecutively and together, and this happens many times for each of them. Figure 27 shows the ten most common bigrams and the number of times they appear twice together within the ciphertext.

Order	Bigram	Appears as	Occurrences
1	JJ	JJJJ	37
2	JM	JMJM	36
3	MJ	MJMJ	30
4	JH	JHJH	23
5	HJ	HJHJ	35
6	MM	MMMM	26
7	MH	MHMH	32
8	JD	JDJD	23
9	JO	JOJO	28
10	HM	HMHM	33

Figure 27. Trigram distances between repetitions.

Being zero the minimum distance between occurrences of the same repeated bigram and having the same happening for all 10 most common bigrams across the ciphertext, allows us to affirm that the distances between repeated occurrences of bigrams gives absolutely no information about the length of the key, making totally useless a Kasiski statistical analysis attack based on the repetition of bigrams.

Second, we took the three most repeated trigrams, MMJ, which repeats 763 times within the ciphertext, HJJ, which repeats 759 times and MJJ, which repeats 755 times.

We built up a table listing the distances from each appearance of each trigram to the next in one column and the list of distances sorted out from the smallest distance to the biggest and we did the same for each of the three trigrams.

For the first trigram (MMJ) only 58 out of the 763 repetitions are at a distance that is a multiple of the key length of 13 characters and only 5 of them were a multiple of the less common multiple of the lengths of the key and the second parameter.

For the second trigram (HJJ) only 48 out of the 759 repetitions are at a distance that is a multiple of the key length of 13 characters and only 2 of them were a multiple of the less common multiple of the lengths of the key and the second parameter.

For the third trigram (MJJ) only 49 out of the 755 repetitions are at a distance that is a multiple of the key length of 13 characters and only 7 of them were a multiple of the less common multiple of the lengths of the key and the second parameter.

It is clear that the encryption generates a diffusion of the results even using short keys, which makes any statistical analysis attack unfeasible. No matter how you compare the results of analyzing the most repeated trigrams, no information about the key length can be obtained.

Figure 28 shows the first 24 results for each of the trigrams.

	MMJ		HJJ		MJJ	
	U/O	O	U/O	O	U/O	O
1	11,339	***5***	771	4	5,020	***7***
2	6,817	***11***	***487***	15	11,054	***11***
3	8,828	12	19,573	38	***4,003***	***13***
4	1,944	15	8,517	55	***6,997***	20
5	6,598	22	1,248	58	5,062	32
6	11,538	45	854	69	4,700	44
7	9,032	46	***13,103***	80	19,637	***53***
8	2,079	66	***2,111***	82	7,679	58
9	33,580	75	2,507	91	6,878	66
10	638	106	692	***109***	1,784	66
11	1,164	121	20,869	***113***	36,239	74
12	11,664	124	5,634	133	8,294	77
13	12,180	130	1,390	***139***	6,008	99
14	1,687	156	6,226	144	6,341	152
15	13,358	171	12,450	160	7,704	***163***
16	4,925	183	287	***227***	2,888	190
17	***2,659***	187	1,323	228	6,840	216
18	2,018	192	5,049	***239***	22,798	***233***
19	1,394	***199***	26,270	***241***	26,708	242
20	26,814	209	42,915	243	2,959	266
21	***4,457***	216	11,013	246	7,174	268
22	3,385	242	***797***	260	***5,527***	***269***
23	10,621	259	***5,021***	267	10,530	270
24	***2,411***	265	5,061	267	9,970	284

Figure 28. Trigram distances between repetitions.

U/O stands for unordered, listing the distance from an occurrence of the trigram to the next one, while O stands for ordered, which is an ordered list of the distances between two occurrences of the same trigram ordered in ascending order from the shortest distance between two occurrences of the same trigram to the longest one.

This table shows that it is not possible to find any relation between the distances of the different trigram repetitions and the key length. We grayed out and styled in bold and italic those distances between two consecutive repetitions of the same trigram that are in fact a prime number, meaning either the key cannot be shorter or they are random repetitions.

It is clear that a Kasiski statistical analysis attack will not succeed even considering that the key is not random and extremely short when compared to the length of the original plaintext.

C. Friedman's Index of Coincidence Attack

The formula to calculate the Index of Coincidence (IC) for any given text is as follow:

$$Ic = \sum_{c \in alphabet} \left(\frac{count(c) \times (count(c) - 1)}{Length \times (length - 1)} \right) \tag{20}$$

Theoretically perfect IC is defined as if all characters occurred the exact same number of times so that none was more likely than any other to be repeated, so, for an alphabet of 26 characters, it was calculated to be 1/26, which is approximately 0.03846. The closest the IC of a given ciphertext is to the perfect IC, the more difficult it will be to try to obtain the key length or the original plaintext.

The calculated IC of the original plaintext is 0.0659 and the calculated IC for the ciphertext produced by our encryption is 0.03874, a difference of 0.00028 from the perfect IC.

Poly-substitution encryption generates a level of diffusion that makes obtaining the key length through the IC an impossible task.

D. Linear Cryptanalysis.

The discovery of linear cryptanalysis is credited to Mitsuru Matsui [36], who first applied the technique to the FEAL cipher in 1992. Later, he published an attack on the Data Encryption Standard (DES) [37].

Linear cryptanalysis focuses on finding a linear relationship between a subset of plaintext bits and a subset of data bits that behaves in a non-random fashion. It is a known-plaintext attack, meaning the attacker will have some sets of plaintexts and associated ciphertexts, all encrypted with the same key. It was first intended as a cryptanalysis attack to DES, but proved to be useful for other multi-round fixed-block ciphers. It requires to have some pairs of known-plaintext/ciphertext pairs encrypted with the same key. The first difficulty linear cryptanalysis will find is that the key does not have a fixed length and also the processing block size or sizes are also of not fixed length. The second difficulty is that there are no s-boxes and the encryption is made in a single round. Even if the attacker gets a huge number of known-plaintext/ciphertext pairs all encrypted using the exact same parameters, the length of the key and the length or lengths of the processing block remain unknown and those pairs of known-plaintext/ciphertext give no clue about them, making unfeasible the use of linear cryptanalysis to attack poly substitution encryption as it is defined in this paper.

E. Differential Cryptanalysis.

Differential cryptanalysis discovery is usually credited to Adi Shamir and Eli Biham, who published a number of attacks against several block ciphers in the late 1980s. Don Coppersmith, a member of the original IBM DES team, published a paper stating that differential cryptanalysis was known to IBM as early as 1974 [38].

Linear cryptanalysis is based on exploiting linear relationships between bits in the cipher, while differential cryptanalysis uses differential relationships between various bits in the cipher. Differential cryptanalysis is a chosen-plaintext attack where the attacker is able to make a cryptosystem encrypt data he chooses using the target key which is unknown and remains secret. Analyzing the ciphertext obtained (which is known), the attacker can obtain the key used.

The standard differential cryptanalysis method is a probabilistic chosen-plaintext attack. It is also oriented to multi-round ciphers like AES with a fixed length key and a fixed length processing block. There is no way the cryptosystem may encrypt without providing all the parameters, including the key and as we have seen in the linear cryptanalysis attack, even having pairs of known-plaintext/ciphertext gives no information about the lengths of the key and the processing block or blocks used, making unfeasible the use of differential linear cryptanalysis to attack poly substitution encryption as it is defined in this paper.

F. Side-channel attacks..

A side-channel attack is an attack based on knowledge gained from the implementation of a computer system instead of weaknesses from the encryption algorithm itself. As the encryption we propose is based in simple arithmetic operations like addition and subtraction of byte values, timing and power-analysis attacks will fail as well as other side-channel attacks like Power-monitoring attacks, electromagnetic attacks or differential fault analysis just because none of them may distinguish an addition from a subtraction or two different additions being performed.

G. Other Modern Cryptanalysis

Time-Space Trade-Offs like Diffie-Hellman's meet-in-the-middle attack, Hellman's Time-Space Trade-off, or Rivest's Distinguished Endpoints just will not work, and we will explain why.

Diffie-Hellman's meet-in-the-middle attack is oriented to break multiple-encryption algorithms repeating the same encryption using different keys as in Double-DES and Triple-DES and it is not the case with our single round poly substitution encryption.

Hellman's Time-Space Trade-off is based on pre-computing sample plaintext/ciphertext pairs using random keys and considering the mapping of key k to ciphertext c as a random permutation function f over an N point space, being N the total number of possible keys and also assumed to be the total number of possible plaintexts and ciphertexts. The first difficulty such approach will face is that there is not an unique random permutation function f. For the same plaintext, it will exist more than one key-substitution pair sequence that will produce the exact same ciphertext. If we consider that we are using an alphabet A with a characters or symbols and we are using a third parameter with the same alphabet and using a number a of different substitutions, for any given plaintext symbol-ciphertext symbol pair and for each different substitution, there will be a key value that will produce the ciphertext symbol from the plaintext one using that substitution and key values. The crucial issue Hellman's Time-Space Trade-off will find with poly-substitution encryption is that if the number of possible plaintexts and ciphertexts is N, and an alphabet with a symbols is used, the number of possible keys will be in the order of a^N, which for $a \geq 2$ is a lot bigger than N, making unfeasible to use this attack.

Rivest's Distinguished Endpoints is an enhancement of Hellman's Time-Space Trade-off but will face the same difficulties with poly-substitution encryption and will not work for the same reasons.

IX. Partial or Limited Data Universes

In certain situations, some byte values or sequences may be considered restricted, invalid or not acceptable. As an example, when transmitting data, the end-of-message value cannot be part of the transmitted message and an encrypted message cannot include the end-of-message within it. Also, some structured messages require the information stored within the message to comply with some structural lengths and data types for specific parts of the message and to encrypt such messages, the encryption must respect data structures and formats, something none of the available encryption solutions, either symmetric or public-key can provide.

A. Format Preserving Encryption

Format Preserving Encryption (FPE) refers to encrypting in such a way that the output ciphertext is in the same format as the input plaintext, including having the exact same length. If the plaintext is just numbers, you get numbers, if it is alphabetic characters you get alphabetic characters, etc.

For example: To encrypt a sixteen-digit credit card number so that the ciphertext is another sixteen-digit number, or, to encrypt a nine-digit social security number so that the ciphertext is another nine-digit number, or to encrypt a person's name so the ciphertext is another alphabetic string with the same length.

One reason to use FPE comes from the difficulty to integrate encryption into existing applications with well-defined data models like banking, industry, financial technologies or medical records databases among others.

Adding encryption to such applications comes with high costs associated to the field length limits, data type changes and computational power required.

Recent scandals like de Panama papers, the NSA secrets offered in auction or the World Anti-Doping Agency (WADA) medical records disclosure are clear examples of the need for format preserving encryption to protect sensitive information within databases. In the Panama papers and the WADA cases, if the fields containing sensitive information would have been encrypted, the information stolen would have been useless for the hackers despite the security breach they achieved, precisely because the names of the persons involved and other personally identifiable information (PII) would have been encrypted.

B. Symmetric Key Format Preserving Encryption

Block ciphers cannot preserve the plaintext length without additional work unless it exactly matches the block size used or an exact multiple of it.

If you are trying to encrypt a nine-digit social security number stored in a nine byte plaintext, by default a block cipher like AES will return a 16 byte (128 bits) ciphertext that cannot be guaranteed to be numeric.

John Black and Philip Rogaway [39] described three ways to implement Format Preserving Encryption they proved as secure as the block cipher used to construct each of them:

- FPE from a prefix cipher.
  Assign a pseudorandom weight to each integer in the range {0,…,N-1} and then sort by weight. The weights are defined by applying a block cipher to each integer and then sorting by the result ciphertext value. A different key will result in a different weight order. The size and number of entries required for the lookup table and the number of encryptions that need to be performed to initialize the table make this technique impractical for large values of N.
- FPE from cycle walking.
  If there is a limited set of valid values within the block cipher permutation domain, a Format Preserving Encryption algorithm can be created by repeatedly applying the block cipher until the result is within the valid ones. As the domain is finite and the permutation is one-to-one, the cycle walking is guaranteed to terminate, but it may end up with the same original value.
  The advantage of this technique is that the valid values do not need to be mapped as a consecutive sequence. The disadvantage is that too many cycles may be required for each operation and the encryption process stops being deterministic as it is impossible to know in advance how much time will the encryption process need.
- FPE from a Feistel network [40][41].
  The output of the block cipher can be used as the source of pseudo-random values for the sub-keys for each round of the Feistel network; the resulting construction is good if enough rounds are used.
  It cannot be guaranteed that the Feistel network will preserve the format, but it is possible to iterate it in the same way as the cycle-walking technique to ensure the format can be preserved.

The United States' National Institute of Standards and Technology (NIST) Special Publication 800-38G [42] defines methods for Format-Preserving Encryption for Block Ciphers that can be used for partial or limited data universes and not limited to numeric values only. The core of the proposed solutions is derived from an approved block cipher with 128-bit blocks, mainly, the Advanced Encryption Standard (AES) algorithm. Classic and currently used Symmetric key FPE requires additional processing which implies more time and computational power is required.

C. *Public-Key Format Preserving Encryption*

There are no developments in public key FPE because based on the math used for the encryption, no secure Public-Key encryption can preserve the length. Besides that, being the key public and known to everybody, an attacker can just try encrypting every possible plaintext to see if the result matches the encrypted text.

If what was encrypted is a Social Security Number (SSN), the attacker only has to try every possible nine-digit numeric value, a mere billion tries that can be accomplished in short time with currently available computational power.

To prevent this, the public key encryption algorithm must not be deterministic and must include some randomness so that a large set of possible ciphertexts may result from a given plaintext using a given public key.

Using RSA as defined by PKCS#1 [43] pads the plaintext with random bytes causing the ciphertext to be necessarily larger than the plaintext.

We can conclude that Public key encryption is not suitable for FPE.

D. *Our Proposed Format Preserving Encryption*

Continuing with our character or byte level data usage, what we need is just an array containing all valid byte values corresponding to the characters or symbols from the defined alphabet, and this array can have at most 256 elements when all possible byte values are valid. The number of elements in the array will be the Module for all module based substitutions.

What each of the valid values represents is irrelevant to the process:

- If what we want to encrypt, preserving the format, is a 16-digit credit card number or a 9-digit social security number, the process needs to have an array with 10 elements with different values, one for each of the decimal values from 0 to 9.
- If what we want to encrypt is a database field corresponding to a person's name and we want the valid values to be all capital letters {A…Z}, all lower case letters {a…z}, the apostrophe and the space, we will need to have an array with 54 elements with different values.
- If what we want to encrypt is a database field corresponding to an address and we want the valid values to be all capital letters {A…Z}, all lower case letters {a…z}, all ten decimal numbers {0…9}, the apostrophe, the comma, the period and the space, we will need to have an array with 66 elements with different values.

Which are those values and how they are ordered is totally irrelevant to the process because we are not using the byte value for the character itself, but its position within the defined alphabet.

E. *A practical example*

Let us suppose an organization wants to encrypt a specific field within a database and the group of valid values for each character in that field, expressed in decimal values are:

{32,39,44,48…57,65…90,97…122}

We can see they are 65 different values, and the group as a whole is non-continuous.

The values can be stored into an array with 65 elements and there can be 65! (8.24765 * 10^90) possible orders for those values.

Please notice there is no mention to the field length. This is because the field id and length are values that can be obtained and used during the process and has no impact over the encryption because everything is encrypted at a byte level.

In this particular example, the module M will be 65 and an encryption algorithm using encryption substitutions like those presented here and complying with the seven steps of our technique will guarantee a very fast single-pass format preserving encryption without the need of any cycle walking or additional processing to obtain a valid result.

It preserves the unconditional security and also offers the advantage that the encryption process timing can be accurately estimated based on the size of the information to be encrypted (or decrypted) independently of its content.

Due to their intrinsic simplicity, all these substitutions are really fast and an algorithm using them and complying with the technique we presented here, can be easily implemented within existing systems without the need of massive investments in computational power or data structures modification.

X. UNCONDITIONAL SECURITY

The plaintext is finite and it is never random, which is a fact. Whatever needs to be encrypted has a measurable length and it is not random. It does not matter whether it is a text, an image, an audio, a video, a blueprint, a spreadsheet or whatever it may be; it is something that can be comprehended. Because if it cannot be comprehended, there is no need to encrypt it.

Our encryption technique does not make any computational assumption and so it does not depend for its effectiveness on any computational hardness.

Vigenère's and Vernam's cipher constraints and requirements for perfect secrecy are due to the use of a single encryption substitution (Modular Addition and XOR, respectively). Given the ciphertext, the only thing that is not known is the key. The encryption substitution order of the Vigenère and the Vernam ciphers are publicly known, it is Modular Addition after Modular Addition and XOR after XOR respectively, repeated as many times as the length of the plaintext.

What happens with the Vigenère and the Vernam ciphers if the key is not random and shorter than the plaintext?

If the key is a sequence of equal characters, it will convert both the Vigenère and the Vernam ciphers into a Caesar cipher that can be easily broken.

If the key is a sequence of alphabetic characters, it will turn Vernam cipher into a running key or Vigenère cipher, which can also be easily broken.

How can we guarantee a cipher based on our technique and complying with the seven steps will offer perfect secrecy?

A sequence of substitutions to be applied to the plaintext and the key can be generated from the third parameter, the substitution pool, the initial substitution order and the length of the key, and this sequence can be generated as long required to match the plaintext length and having no repetitions. That sequence of substitutions will also depend on the plaintext, so a different plaintext will generate a different substitution sequence for the exact same remaining parameters.

So far, we can say that for any single byte, the ciphertext value will depend on the plaintext value, the key value and the specific substitution used.

Being:

- p the plaintext value
- k the key value
- a() the initial substitution order array
- s the third parameter value
- a(s) the value stored in the sth element of the array
- $f_{a(s)}$ the encryption substitution triggered by a(s)
- c the ciphertext value

Equation (21) represents the encryption as:

$$c = f_{a(s)}(p,k) \tag{21}$$

Equation (22) represents the decryption as:

$$p = f'_{a(s)}(c,k) \tag{22}$$

Considering substitution $f'_{a(s)}$ to be the reverse of substitution $f_{a(s)}$.

What happens if the key is shorter than the plaintext and it is not random?

It does not affect the unconditional security because the key does not have a direct impact on the ciphertext that can be inferred in any way. Even if the attacker manages to know the substitution pool, it offers no information about its usage.

The unconditional security is guaranteed based on that for the attacker:

- The key is unknown.
- The key length is unknown.
- The third parameter is unknown.
- The third parameter length is unknown.
- The initial substitution order is unknown.
- The processing block size or sizes is unknown.
- Which ciphertext byte corresponds to which plaintext byte is also unknown.

Even if the attacker knows the substitution pool and has infinite computational power and is able to try all possible keys and all possible substitutions for each byte and every permutation of the results and can purge all the invalid results in just a fraction of a second, the plaintext will still remain hidden at plain sight in a sea of false positives and the attacker will still be unable to decide which elephant on the beach is the right one because every possibly valid plaintext with the same length or shorter has the exact same probability of being the original plaintext without any indication of which one is the right one. Any original text may be padded adding spaces at the end without affecting the text but generating a longer plaintext. Once decrypted, those extra spaces at the end has no impact at all in the text content and meaning.

XI. Practical Applicability

Modern complexity-based cryptography requires to have a large amount of resources available, specifically computational power, processing speed and memory and as the complexity increases resource requirements also increase.

While symmetric ciphers like AES plans to use 512 bit keys and public key ciphers like RSA plan to use 4,096 bit keys in an attempt to resist a quantum attack, as the number of qubits keeps growing so will do their key length requirements. Soon Megabit-long keys will be used and expanded to Gigabit-long keys to later be expanded to Terabit-long keys, and so on, and the same will happen with the computational power and processing speed requirements.

On the other hand, our proposed encryption does not need neither long keys nor high processing speed and will not need to expand key lengths or processing speed requirements as the number of qubits in quantum processors keeps growing. It can be used even with pen and paper and some spare time.

A. Performance Requirements

Although more complex substitutions can be built up, increasing the total number of substitutions available, our example has shown how using basic mathematical operations like modular addition and modular subtraction suffice to provide unconditional security. It is not even required to use slightly more elaborated mathematical operations like multiplication or division, less to use more complex or advanced math.

Less processing power required means simpler, smaller processors emitting less heat and requiring less electrical power consumption. Most hardware controllers from the simplest to the most complex ones already have built-in basic math operations like addition and subtraction embedded making it very simple to add encryption to them without increasing their power requirements.

Considering the computational power required by AES or RSA, what is required for elliptic curve and what may be required for lattices encryption, it becomes obvious that our proposed encryption have much less computational power requirements.

The use of such simple math guarantees true and absolute cross platform encryption/decryption.

These low power requirements allow for this encryption to be easily added to any device either through a hardware, software or mixed implementation at a very low cost without jeopardizing its security.

B. *Memory Requirements*

In our examples, we have shown how the plaintext, the key, the third parameter and the ciphertext can be processed just one byte at a time, when we use a single processing block size of one byte, only the fourth parameter requires a maximum of 256 bytes (2,048 bits) of memory to store the substitution order array when the alphabet used contains all possible byte values.

The minimum memory requirements for a cipher based on our technique will depend on how it is implemented, the substitutions used, the maximum processing block size allowed and the reading and writing buffer sizes.

We have seen that the plaintext can be anything, as far it is a finite sequence of bytes and the same applies to the key and the third parameters, and so the ciphertext will also be a finite sequence of bytes.

C. *Applications*

The list of possible applications is endless, so we will provide just a few of them we are currently working on.

- Encrypt Data at Rest
  We tried our encryption test software in Microsoft Windows encrypting files of different sizes having encrypted and decrypted files up to 1 TB (one Terabyte) without any kind of hassle and with zero errors.
- Encrypt Data in Transit
  Our encryption test software is capable of reading a local file and remotely writing the encrypted file without sending any unencrypted data.
- Encrypted Remote Control
  We are currently testing remote controlling a drone using encrypted control packets and making the drone to identify and ignore any invalid packet making its control hacking-proof.

This is a work in progress and there is still a lot of work ahead before it could be considered complete.

D. *Pros and Cons*

The pros can be resumed in the fact that our proposed encryption is light, fast and unconditionally secure. Other pros are that it also allows for multiple different ciphers to be built up based on it. Its low memory and processing power requirements makes it an ideal solution to add encryption to the Internet of Things (IoT) and all the smart devices it is bringing up. Its Format Preserving Encryption capabilities makes it an ideal tool to develop database encryption solutions that would not require to modify the existing data structures. The best pro is that anything encrypted using a cipher based on it will remain impervious to any quantum attack, no matter how many qubits the quantum processor may have, or whatever may come later.

Due to its simplicity, low requirements and implementation ease, it has no cons, no drawbacks and no special requirements of any kind.

XII. Key and Message Distribution

Safe message and key distribution have been an issue since the very origin of cryptography and have played a major role in the development of the field, but the advent of internet has changed everything. A message of any size can be accurately sent from one point to another through cyberspace, the availability of cloud storage and file repositories has made internet the home of trillions of files of every possible type.

Now, there is no need to even send the encrypted file or the key to the addressee, the address from where they can be downloaded will suffice. The internet address of a file can be much shorter than the file itself. With all the cloud and data files storage providers available around the world, how could a single file be located without knowing its exact name and location?

From cloud storage providers to file storage providers, one single file hidden between trillions of files can be accessed only knowing how to reach it.

Some storage providers and file address link shrinking services allow to shorten the full file name and address to just 7 or 8 alphanumeric characters, each capable of identifying about 3 trillion different files.

The sender and the addressee only need to agree on the storage or storages and exchange only those very short codes already encrypted. A 10 GB file can be shared using an encrypted message containing only 7 or 8 characters that can be anywhere. As part of a comment on a news or blog page, as part of a tweet and an incredibly huge etcetera.

XIII. Conclusions

All cryptography in use is vulnerable to an attacker with enough computational power. Everything that has been digitally stored or transmitted or will ever be digitally stored or transmitted by any means (network, wireless, internet, etc.) may be publicly disclosed sooner than later. Unencrypted databases may be hacked and its content made public like what happened in August 2016 when the World Anti-Doping Agency (WADA) was hacked by Russian hackers and private health records from famous athletes were made public to distress and discredit them.

The simplicity and ease of implementation of poly-substitution encryption sheds a light for the development of true cross-platform encryption solutions and add-ons fully compatible across hardware and software platforms and operating systems.

Poly-substitution encryption may be easily added to existing hardware, software and mixed solutions. Poly-substitution encryption cannot prevent hacking or system intrusions but may make the effort fruitless and useless.

A low cost and unconditionally secure format preserving encryption is urgently needed to preserve the privacy of sensitive but personal information. We want to help protect the privacy of personal information around the world.

Assuming there is currently enough available computational power to try in a very short time every single key length and value, with every single processing block size and every single possible encryption substitution there will still not be possible to decide which one of the apparently valid results is the true original plaintext.

Even knowing that the plaintext is just plain text, any possible text with the same length or shorter (just filled with spaces at the end, at the beginning, or within, in order to reach the same length) has the exact same possibility of being the original plaintext. That is the essence of unconditional security, something none of the currently in use encryption standards or solutions can offer.

We have seen here that this poly-substitution encryption technique offers the same level of unconditional security guaranteed by the Vernam cipher without its constraints.

With billions and billions of files available through the internet and the capability of using any of them as a key, as a third parameter and even as the original substitution order, nobody needs to remember long keys, just needs to remember which files were used and how to reach them.

If one has enough computational power like quantum computing promises to offer when it becomes widely available, one may be able to break and read any file encrypted with any of the current standards, techniques and tools with two exceptions:

- Anything protected through a One-Time-Pad (or a proper use of Vernam's or Vigenère's cipher) will remain secret.
- Anything protected through the use of a cipher based on our proposed technique and complying with its seven steps will remain secret.

XIV. Future Work

As we stated before, this is a work in progress and there is still a lot of work ahead before it could be considered complete.

We have already implemented an encryption solution complying with the seven steps defined here and used it for our tests. It is a Windows app programmed in Visual Basic 6.0 that uses 256 different encryption substitutions and is capable of encrypting and decrypting any kind of files up to about 900 TB (900,000,000,000,000 bytes long) and fast enough to cipher/decipher an 80 MB file in less than five seconds. There is plenty of room to enhance and improve the encryption speed by optimizing the code and using programming languages that may run faster.

We do not have the resources to test up the maximum possible file size but already tested it on a 1 TB (one Terabyte) text file, encrypting and decrypting it without any issue or error. We will continue testing encrypting and decrypting larger files of different types, including databases, audio and video files, images and compressed files, etc.

Future work will aim to validate the ideas presented in this paper by means of additional practical results, simulations, statistical analysis and practical performance comparisons with other ciphers.

We are open to share our development with the cryptographic community to be fully analyzed, tested, improved and enhanced.

Future work will also aim to develop and test practical solutions for low cost Format Preserving Encryption algorithms based on the technique presented here.

References

[1] J. Murguia Hughes, "Seven Steps to a Quantum-Resistant Cipher", SECURWARE 2016, The Tenth International Conference on Emerging Security Information, Systems and Technologies, pp. 247-253, ISSN 2162-2116, ISBN 978-1-61208-493-0.

[2] W. Whitfield and M. E. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory. 22 (6): 644–654.

[3] H. Sidhpurwala, "A Brief History of Cryptography" redhat Security Blog, August 14th, 2013.

[4] G. B. Belasso, "La cifra del Sig. Giova Battista Belasso", 1553.

[5] G. S. Vernam, "Cipher Printing Telegraph Systems for Secret Wire and Radio Communications", Journal of the IEEE 55: 109-115.

[6] G. S. Vernam, Patent 1,310,719. "Secret Signaling System", Patented July 22, 1919. United States Patent and Trademark Office.

[7] C. E. Shannon, "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. XXVII, No. 3, July 1948, pp. 379-423 and October 1948, pp. 623-656.

[8] C. E. Shannon, "Communication Theory of Secrecy Systems", The Bell System Technical Journal, Vol. XXVIII, No. 4, pp. 656-715.

[9] F. W. Kasiski, "Die Geheimschriften und die Dechiffrierkunst", 1863.

[10] Swenson, C: "Modern Cryptanalysis - Techniques for Advanced Code Breaking", John Wiley & Sons Inc. 2008.

[11] Friedman, W. F.: "The index of coincidence and its applications in Cryptanalysis", Cryptographic Series, 1922.

[12] ETSI, "Quantum Safe Cryptography and Security", ETSI Whitepaper No. 8, June 2015, ISBN No. 979-10-92620-03-0.

[13] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", SIAM J. Comput. 26 (5): 1484–1509.

[14] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)", FIPS PUB 197, United States National Institute of Standards and Technology (NIST), November 26, 2001.

[15] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)", Fast Software Encryption, Cambridge Security Workshop Proceedings (Springer-Verlag): 191-204, 1993.

[16] S. Lurye, "An uncertain path to quantum supremacy: Notes from RSA", Kaspersky Lab Daily, May 8, 2018, https://www.kaspersky.com/blog/quantum-supremacy-rsa/22339/. 2018.05.31

[17] J. A. Buchmann, D. Butin, F. Göpfert and A. Petzoldt, "Post-Quantum Cryptography: State of the Art", Springer LNCS, volume 9100: 88–108.

[18] L. Carrol, "Through the Looking-Glass", Macmillan, 1871.

[19] "Data Encryption Standard (DES)", FIPS PUB 46, United States National Institute of Standards and Technology (NIST), January 15, 1977.

[20] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". Journal of statistical physics. 22 (5): 563–591. 1980.

[21] Y. I. Manin, "Vychislimoe i nevychislimoe [Computable and Noncomputable]" (in Russian). Sov.Radio. pp. 13–15. 1980.

[22] R. P. Feynman, "Simulating physics with computers". International Journal of Theoretical Physics. 21 (6): 467–488. 1982.

[23] D. Deutsch, "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer". Proceedings of the Royal Society of London A. 400 (1818): 97–117. 1985.

[24] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, "Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits", arXiv:1710.05867 [quant-ph].

[25] W. Knight, "IBM Raises the Bar with a 50-Qubit Quantum Computer", MIT Technology Review, November 10, 2017, https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/. 2018.05.16

[26] D-Wave Systems, The Quantum Computing Company. http://www.dwavesys.com/. 2018.05.16.

[27] "Not Magic, Quantum", 1663 - The Los Alamos Science and Technology Magazine, July 2016, pp. 14-19.

[28] B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich and W. K. Hensinger, "Blueprint for a microwave trapped ion quantum computer", Science Advances, 01 Feb 2017,Vol. 3, no. 2, e1601540.

[29] The Oxford Math Center from Oxford College of Emory University, "Letter Frequencies in English", http://www.oxfordmathcenter.com/drupal7/node/353. 2018.06.07

[30] Wikipedia, "Frequency Analysis", https://en.wikipedia.org/wiki/Frequency_analysis. 2018.05.16.

[31] Project Gutemberg, "Project Gutenberg Complete Works of Winston Churchill by Winston Churchill", https://www.gutenberg.org/ebooks/5400. 2018.05.16.

[32] CrypTool-Online, https://www.cryptool.org/en/, 2018.05.16.

[33] E. Barker and N. Mouha, "NIST Special Publication 800-67 Revision 2: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", National Institute of Standards and Technology (NIST).

[34] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)", Fast Software Encryption, Cambridge Security Workshop Proceedings (Springer-Verlag): 191-204, 1993.

[35] Rl L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems". Communications of the ACM. 21 (2): 120–126. 1978. ISSN 0001-0782.

[36] M. Matsui and A. Yamagishi, "A new method for known plaintext attack on FEAL cipher", Advances in Cryptology – EUROCRYPT 1992.

[37] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard", Advances in Cryptology – CRYPTO 1994.

[38] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks", IBM Journal of Research and Development. Vol38 No. 3 May 1994, pp. 243–250.

[39] J. Black and P. Rogaway, "Ciphers with Arbitrary Domains", Proceedings RSA-CT, 2002, pp. 114–130.

[40] H. Feistel, "Cryptography and Computer Privacy", Scientific American, Vol. 228, No. 5, 1973.

[41] A. J. Menezes and P. C. Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography (Fifth ed.)", 2001, p. 251.

[42] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption", NIST Special Publication 800-38G, United States National Institute of Standards and Technology (NIST), March 2016.

[43] RSA Laboratories, "PKCS#1; RSA Cryptography Standard", RSA Laboratories.