Seven Steps to a Quantum-Resistant Cipher

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

***Abstract*—All cryptography currently in use is vulnerable and will become obsolete once quantum computing becomes 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 share advances in searching for perfect secrecy instead of complexity and we try to light a path to a whole new quantum-resistant cryptography.**

Keywords-cipher; quantum-resistant; cryptography; secrecy; 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.

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

Although the first known evidence of some form of cryptography is almost four millennia old [1], one of the oldest known form 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 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. 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.

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 cipher text. 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.

It has been said and repeated that quantum computing will make obsolete all existing cryptography because it will allow brute force attacks to be completed in a short period of time.

All existing cryptography? No, there is an exception.

About a century ago, Gilbert Vernam invented an encryption technique [2] (Patent US 1310719 [3]) that thirty-something years later Claude Shannon proved [4] it was unbreakable and will remain unbreakable to quantum computing. It is not used because it requires the key to have the same length as the plaintext, to be random and not to be reused.

As today's information is always measured in bytes or multiple of bytes (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 be a zero (0) or a one (1).

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

For a byte, the Vernam cipher will use the XOR function between the plaintext byte and the key byte. The function will compare each bit within the first byte to the bit in the same position in the second byte and will generate a bit with a value of zero if both bits have the same value and one if they are different. The function will return the cryptogram byte as its result. For a specific plaintext byte value, each of the 256 possible values of the key will produce a different cryptogram byte value.

If you get the cryptogram 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) 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 [5].

In this paper, we will present our proposed encryption technique (patent pending [8]) and the seven steps to an unbreakable quantum-resistant cryptographic technique.

The rest of this paper is organized as follows. Section II describes the state of the art and the vulnerability to quantum attacks. Section III describes each of the seven steps of our proposed encryption technique. Section IV describes the analysis of a possible cipher based on those seven steps and compares it against Vernam's and other current standards. Section V describes the conclusions and Section VI describes the future work and goals.

II. State of the Art

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" [6].

Discussion and comparison between symmetric and public key cryptography currently in use becomes irrelevant once one understands that none of them is unbreakable.

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 [7] and so, they are deemed to be insecure to quantum computing.

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

Any cipher that bases its strength on its complexity and 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.

Vernam's cipher and the one described in this paper make no computational assumptions and are both information-theoretically secure.

What we hope to achieve is to provide a cipher offering perfect unconditional security against eavesdroppers no matter how arbitrarily powerful they may be or become in the future and without the constraints the Vernam cipher has. Something none of the currently in use standards can offer.

III. The Seven Steps

A. *Step One (Use Multiple Encryption Functions)*

Vernam used a single function (XOR). Our approach will use many of them. Each function will take the plaintext byte and the key byte and will return a cryptogram byte and for each of the 256 possible key byte values will return a different cryptogram byte value.

Below, we will explain two of these functions that are similar and as unbreakable as the Vernam or XOR function; other functions with the same behavior will also be unbreakable.

- **Modular Addition:** Will add up the plaintext byte value and the key byte value wrapping up at 255. If the result is larger than 255 it will subtract 256 from the result. For a specific plaintext byte value, each possible key byte value will produce a different cryptogram byte value. If you get the cryptogram 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.
- **Modular Subtraction:** Will subtract the key byte value from the plaintext byte value wrapping up at zero. If the result is negative it will add 256 to the result. For a specific plaintext byte value, each possible key byte value will produce a different cryptogram byte value. If you get the cryptogram 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.

Using multiple functions provides additional security because, if one has the cryptogram byte, not only the key byte used is unknown, but also the function used.

For a given plaintext byte value, any valid function should return 256 different results based on the value of the key byte, so applying each function to the given plaintext byte value and for each key byte value from 0 to 255, will produce a list of 256 different results.

Each of those functions is as secure and unbreakable as Vernam's XOR and using many of them does not diminish either the security or the unbreakability of the process.

When the plaintext's length is larger than one byte we can use one function to process the first byte, another one to process the second byte and so on. That leads us to the following step.

B. *Step Two (Use a Second Parameter)*

A second parameter will be used to indicate which function to use on each instance.

A block from this second parameter will indicate which one of the many available functions will be used to process a byte from the plaintext and a byte from the key.

Let us say we decide to use only 256 different functions from all that can be created. In such case, we will only need one byte from this second parameter to indicate which of those functions will be used for this specific plaintext byte and key byte.

So far, the second parameter byte value x will trigger function z.

How do we know which of the available functions is function z, is explained in the next step.

C. *Step Three (Order of the Functions)*

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

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

And this list is not unique, 256 different functions can be ordered in 256! (n! = 1x2x3x,…n) different ways (256! is a 507 digit decimal number with a value larger than 8.578 * 10^506 or about 2^1684), and a different function order will produce a different cryptogram for the same plaintext and key.

Now, an attacker not only needs to try every possible key, also needs to guess which functions were used and which function is triggered by each possible value of the second parameter. And that, assuming the selected function order is hardcoded within the process.

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

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

D. *Step Four (Changing the Order of the Functions)*

How do we make second parameter byte value x to trigger a function different from function z?

The solution is both simple and elegant.

We add a third parameter. One of those 256! possible orders of the numbers from 0 to 255 is loaded into a 256 elements vector, and value x is used to point to the vector's element whose value will be used to trigger the function.

A different third parameter will provide a different function order.

As this third parameter is a sequence of 256 values, each between 0 and 255, it is possible to exclude certain values just by replacing them (i.e., if you want the value 14 not to be used, then replace the element with a value of 14 for a different value).

Now, second parameter byte value x will trigger a function depending on the x^{th} element of the third parameter.

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

Next step will show how to change that.

E. *Step Five (Block Processing)*

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

The last block may be shorter but it is equally processed from last byte to first one as any other block without any need of any 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 the original plaintext.

F. *Step Six (Key Length and Key Repetitions)*

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

Vernam's cipher requires the key to have at least the same length as the plaintext. If the key is shorter, the process starts to repeat and it weakens its security.

If we use a key shorter than the plaintext it will wrap up at the end, but unless the key and the second parameter both have the exact same length, there will be no repetitions until we reach a position within the plaintext equal to the minimum common multiple of the lengths of both the key and the second parameter. And 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, before starting to repeat it, the process changes the function order by modifying the elements in the vector explained in step four.

Each time this happens, the change process behaves differently.

Now, even if the key and the second parameter have the exact same length and they start to repeat in the exact same order, the sequence of functions triggered will not be the same and so no repetitions will occur.

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 second parameter and the processing block.

The key length may go from a single byte to any length, even the same length of the plaintext or longer.

The second parameter may go from a single byte to any length, even the same length of the plaintext or longer.

The processing block size may go from a single byte 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.

When building up the application, different groups and number of functions may be used to create personalized non-standard versions.

IV. ANALYSIS

A. A cipher complying with these seven steps

If we build up a cipher complying with these seven steps, it may use up to four parameters:

- The key to be used.
  This key is just a sequence of bytes of any length and can be longer, equal in length or shorter than the plaintext.
- A second parameter defining which function to use on each instance.
  This second parameter is a sequence of bytes of any length and there is no relation between its length and the lengths of the plaintext or the key.
- An original function order.
  This is a 256 bytes string that will be used to define an initial order for the encryption functions to be used.
- A processing block size.
  This will define the number of bytes to be read at once from the plaintext and processed in reverse order (from the last byte to the first one) to generate the cipher text. A value of 1 (one) will make the plaintext to be processed straight from the first byte to the last one.

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 the plaintext to process, the key, the second parameter, the initial function order and the processing block size.
2. The process loads the initial function order into a 256 element vector.
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 byte from the processing block.
6. The process takes a byte from the key.
   If the key has been exhausted, reorder the original function order vector elements and read the first key byte again.
7. The process takes a byte from the second parameter.
   If the second parameter has been exhausted, start over from its first byte.
8. The process uses the byte from the second parameter to point to an element from the initial function order vector and uses its value to trigger an encryption function passing the plaintext and key bytes as parameters.
9. The function triggered returns a cipher text byte that is written to the cipher text output.
10. The process takes the previous byte 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 cipher text instead of the plaintext and reversing the encryption process.

B. Comparing this cipher with Vernam's one

Is easy to see that if one of the possible encryption functions used is the XOR function and the initial function order vector elements all have the same value and that specific value triggers the XOR function, then and only then, this cipher will behave the same as the Vernam's one.

A text message properly ciphered through the Vernam Cipher 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.

The Vernam Cipher is not used because it has 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 Vernam used a single encryption function (XOR) in the process.

With the Vernam Cipher, for any given cipher text byte, one needs to try any possible key byte value and you will end up with 256 different results, each one with the exact same probability of being the plain text byte value.

With our proposed encryption technique and even assuming the attacker knows the exact processing block size used for this specific cipher text, all the encryption functions used and can match each cipher text 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 functions involved. So, if we used 256 different encryption functions, the attacker will end up with 65,536 possible values for the plain text 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 cipher text, the second one to any of the bytes except the last and so on, doing the math it means there are n! (n! = 1x2x3x,…n) possible orders for the cipher text to match the byte order of the plaintext, being n the length in bytes of both the plaintext and the cipher text.

Our proposed cipher does not have the same constraints as the Vernam one.

Figure 1 shows a comparison between Vernam's cipher and our proposed one:

	VERNAM	Our Proposed Cipher
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 functions	1	256
No. of possible results per function	256	256
No. of possible results per Byte	256	65536
Ciphertext to plaintext match	1	140!
No. of possible results per Byte from brute force attack	256	65536 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 1. Comparing Vernam's cipher to our proposal.

The key may have any length and it does not matter if it is shorter than the plaintext because we can assure the same key value-encryption function sequence will not be repeated.

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.

What if the key is reused?

Using the same key again is irrelevant as far as we do not use the same processing block size, same second parameter and same initial function order altogether again.

C. Comparing this cipher with currently used ciphers

Due to their extreme complexity, none of the current encryption standards will produce a false positive when a wrong key is used and that is why they are vulnerable to brute force attacks.

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 fail 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, second parameter, initial function order or function set; 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.

Figure 2 shows a comparison between our proposed cipher and other symmetric ciphers:

	Block Size	Key Size	Rounds	False Positives
DES [9]	64 bit	56 bit	16	No
3DES [10]	64 bit	128 bit	48	No
AES [11]	128 bit	128, 192, 256 bit	9, 11, 13	No
BLOWFISH [12]	64 bit	32-448 bit	16	No
Our Proposed Cipher	Variable	Variable	1	Yes

Figure 2. Comparison between this cipher and current symmetric standards.

Public-key cryptography currently in use (including RSA and ECC) relies on the presumption 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.

Picture 3 shows public-key key sizes broken in recent years and projections for the near future:

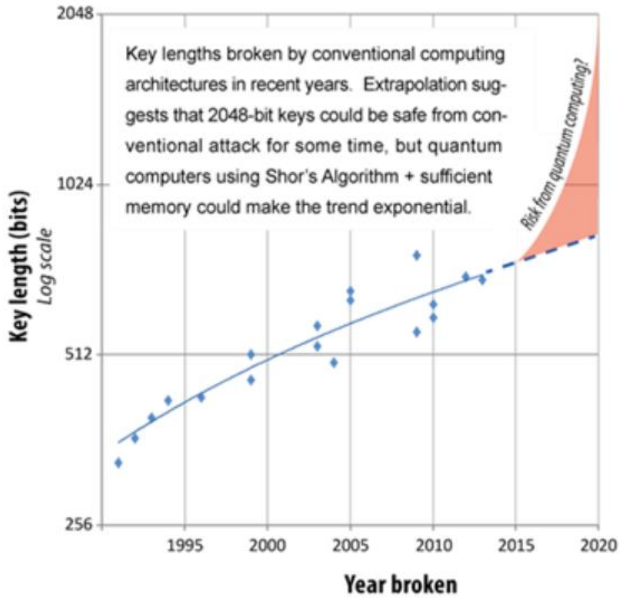

Figure 3. Breaks of the RSA cryptosystem in recent years using conventional computation [6].

D. *Attacking this cipher*

Trying to retrieve the plaintext from a cipher text created through an implementation of this cipher without having any additional information will be at least as difficult as trying to retrieve the plaintext from a cipher text created through a proper use of the Vernam cipher having only the cipher text.

Any attack must take into consideration that all the parameters are external to the process and they all may be different from one plaintext encrypted 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.

Any possible attacker will need to face the following difficulties when attempting a brute force attack to break an encryption created with an implementation of this cipher:

- Which cipher text byte corresponds to each plain text byte.
- Which encryption functions exist and which of them were used.
- Which function was used on each instance.
- Which was the key used.

Let us give the attacker the advantage of knowing all the encryption functions involved, the specific set used to create the cipher text and the processing block size used. In such situation, for each byte in the cipher text, the attacker needs to try every possible function for every possible key byte value and so, instead of getting 256 possible values as with Vernam's cipher, the result will be 65536 possible values having every single one the exact same probability of being the plaintext byte value despite the repetitions.

And that is the best case scenario for the attacker.

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

V. CONCLUSIONS

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 function there still will 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 to reach the same length) has the exact same possibility of being the original plaintext. And that is the essence of perfect secrecy, something none of the currently in use encryption standards or solutions can offer.

We've seen here that this new encryption technique offers the same level of perfect secrecy 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 second parameter and even as the original function 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 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 the proper use of the Vernam cipher will remain secret.
- Anything protected through the use of our proposed cipher will remain secret.

VI. FUTURE WORK

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. It is a Windows app programmed in Visual Basic 6.0 that uses 256 different encryption functions and is capable of encrypting and decrypting files up to about 900 TB (900,000,000,000,000 bytes long) and fast enough to cipher/decipher a 40 MB file in less than five seconds.

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

Future work will also aim to test and evaluate the ideas presented in this paper and their application for Format Preserving Encryption.

REFERENCES

[1] H. Sidhpurwala, “A Brief History of Cryptography” redhat Security Blog, August 14th, 2013.

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

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

[4] 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.

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

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

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

[8] J. Murguia Hughes, “Poly-Algorithmic Encryption Technique”, Patent Pending.

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

[10] W. C. Barker and E. Barker, “NIST Special Publication 800-67 Revision 1: Recommendation for the Triple Data Encryption Algorithm (TDEA)”, NIST Special Publication 800-67, January 2012.

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

[12] 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