Introduction

With the invention of the internet the ways we communicate, do business and gather information has changed within a few decades. Personal data, business contracts, documents with ideas that might lead to patents or intellectually property rights, medical records, financial statements and legal documents were send via the net and we hoped that they were protected and safe from prying eyes. Governments and email providers assured us that the safety of our data was important to them and companies selling encryption software assured us that their products and the encrypted code they created could not be broken and we believed them. That was until the day, the revelations by Edward Snowden made the headline news.

After that we knew that governments had their own agendas as far as our privacy and the safety of our communications was concerned and that constitutions and laws that had been put in place to protect us and our privacy became meaningless. National Security was declared the overriding factor that permitted governments and their agencies to circumvent these laws, backed by secret courts that had been installed, with the majority of the public not even knowing that they existed. Nobody bothered to explain to the public what National Security actually stands for and the rubber blanket created by our politicians can stretch above everything and cover anything they feel the public doesn't need to know.

We have reached the point where even discussing issues like this, might place us under suspicion by state agencies to be labelled and placed into the same corners they reserve for anarchists, terrorists and other criminal fraternities. Our private discussions have been reduced to locations where public surveillance and the internet don't have access. The most common slogan, *if you have done nothing wrong, you have nothing to fea*r has been proven wrong in the past. We might say or write something today, which we might regret tomorrow; because it has been intercepted, misinterpreted, and archived only to be used against us when the political wind in the country has changed.

So we believe that encryption of our emails, documents etc is a priority and it should be a form of encryption that ensures that what is private or confidential stays private and confidential. If the state wants to know what we wrote, it has to follow the legal process and ask us for the key to unlock our communications. The request should be supported by evidence that raises suspicion against us and a simple '*We want it because we can't break your encryption!*' isn't good enough. One of the top position in the new US administration in January next year has been linked to a senator who publicly stated that using private encryption also makes one suspicious; a statement that one expects in a totalitarian country but certainly not in a democracy that is regarded as the beacon of democratic rules and laws; and if the future President of the US publicly asks a foreign secret service to hack into the emails of his opponent during his election campaign it should not only raise an eyebrow but set off all alarm bells.

In the following paper we will concentrate on the OTP (One-Time Pad) and a way to avoid some of the pitfalls that come with the original OTP and taint it, not really to be a useful tool for encryption in our times. We will remove the need for the second secure key transmission, which is the main obstacle when the OTP is mentioned. Readers that are not familiar with encryption can read a very useful and entertaining article written by ***Barry K. Shelton***, a US patent lawyer and partner in a US law firm, who also has a strong interest in software and cryptology. The article tells about the need for encryption; the past and present of these tools and how symmetric and asymmetric encryption work.

http://www.infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

We would like to thank *Barry* for permitting us to quote from his article and to display the link here.

The One-Time Pad

*T*he One-Time Pad, you might love it, hate it or being indifferent towards it, is an encryption tool, which after its conception at the end of the 19th century or the beginning of the 20th century (*depending which account we take for true*) hasn't changed at all in the way it operates. A search on the internet will reveal millions of articles dealing with it but all of them point to the fundamental flaw it carries, which is the *key* (*or keys, which is a matter of interpretation*) that has to be shared between sender and recipient and makes it, to put it mildly, not a real contender compared with the systems that are based on mathematical algorithms. The amount of data that needs to be exchanged, and the secure channel the *key transmission* requires, which is of the same size as the plaintext message are regarded as an obstacle as far as security and speed of communications go. One starts to wonder why something invented by men couldn't be improved over the years and instead a label has been placed on it that puts it into the realm of cosmic laws like the speed of light or gravity; permitting no changes to be applied to the modus it operates.

*P*reviously on our website we wrote about the OTP (*one-time pad*) and suggested solutions, offering a different modus operandi (*method of operation*). One of the solutions was based on permutations and the second one using permutations and the difference of positions characters hold in these permutations. Both solutions offered secrecy and security and would require a brute-force attack and a tremendous amount of time and computing resources an adversary would have to invest to break the ciphers these two options offered. We had placed our ideas into the public domain to get a response how readers feel about the OTP and to receive responses, prior to publish our final paper, and that would enable us to write about and address the main concerns raised. We would like to thank the readers that stood with us during the course and commented on our ideas, with their comments and in one way or the other influenced our writing. They have been taken into account and incorporated into this paper, ensuring that everybody without an in-depth knowledge of mathematics or cryptology will understand what we are putting forward here. So without further delay let us jump into the subject.

*T*he original OTP requires a message called the plaintext and for each character the plaintext contains, a *key* character is selected, preferably by a human operator to ensure randomness, even distribution and to avoid that ***the key or part of the key*** is repeated. Now modular arithmetic is applied to turn plaintext and *key* into a cipher. (*Readers should keep this part in mind when we come to the system we developed and define what we understand to be a key when looking at the OTP. Here we went ahead with the semantics used in most publications that describe the OTP.*)

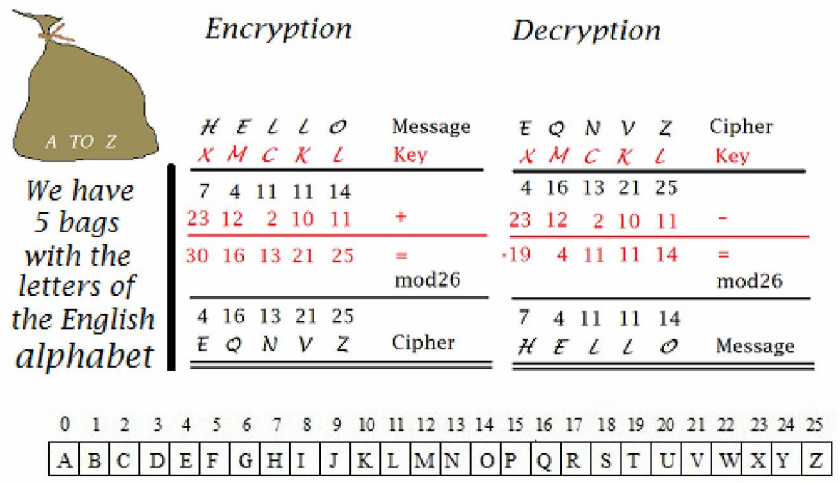


Image 1 - OTP

A recipient now requires both, *cipher* and *key* to reverse the process to read our message. During and after WWII the OTP was one of the favourite tools to carry secret messages across borders. *Key codes* pre-prepared were carried by agents across borders and they only had to wait for a ciphered message to reverse it back into the plaintext. Intercepting the cipher didn't help since without the *key* it was useless for the party that intercepted it.

*N*ow let us take a look at the procedure how the key is generated. An operator has to pick for each plaintext character a random key character and uniting the key character with each plaintext character, meaning that each plaintext character and its key character and the values they represent on an alphabet (*in our case the English alphabet* – A *value 0 to* Z *value 25* – 0 *to* 25) are added together using modular addition (*mod26 – see Image 1 – the value becomes our cipher value)*. If a number exceeds 26, then the value after subtraction of 26 becomes our cipher value. The rule is that if we pass the letter Z, we start again at A. The cipher values are now taken and replaced with the letters they represent on our alphabet string A to Z (*0 to 25*). We have so far now key and cipher, but the problem we created is how to transmit both to the recipient without both being intercepted by an adversary. Intercepting one wouldn't help an adversary but intercepting both would give the plaintext away. This is called by *Bruce Schneier* the *Key Distribution Problem*. It isn't a problem for the system (*which is an OTP*) we developed and suggest in this paper.

*T*he first question we have to ask ourselves is if we have one unique key for the full plaintext or if each character is an independent key in its own right? We maintain that in the example (*Image 1 - OTP*) we have five individual keys. Our reasoning is straight forward because, if a key would be unique and apply for the entire plaintext a missing part of the key should not permit to gain any knowledge of the plaintext if cipher and key are intercepted and only a complete key should permit this. For example if we try to unlock the front door of our house with a key that is only 99% compatible with the original key the door will not open. It will not allow us access to 99% of the house it will keep it locked. Is it import one might ask and the answer yes it is when we come to outline our system.

*T*he second question we have to ask ourselves is if the OTP is really that kind of a special case in cryptology or if that is only a myth because nobody took the head out of the box and looked at systems that bear a similarity with it? We will take a look at a permutation cipher and find out if we can discover these similarities.

Key k is the mapping from Plaintext Alphabet to Ciphertext Alphabet
So if m = "HELLO"; k is mapping in Table 1; c = "EQNNZ"

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

Image 2

*T*he first observation is that our cipher is nearly the same as in our OTP example and that only the double character L shows a difference when encrypted by providing the same cipher character for both L characters in the plaintext. From here we can conclude that as long as we encrypt different plaintext characters and an adversary doesn't have the knowledge that we use a permutation of the alphabet we have the same perfect secrecy as an OTP has to offer. On the other hand if an adversary has knowledge and knows that we use a permutation we have destroyed that secrecy. The reason is simple since a permutation cipher that only encrypts one character ($\ell \geq 1$) offers always the probability of 1/26 (English alphabet). By encrypting more than one character using the same permutation ($\ell \geq 2$) we reduce the probability by ***one*** with each character we encrypt and as seen above in *Image* 2, double characters are very easy to spot. On

the next page we will have a look at a procedure that uses for each single plaintext character a different permutation when creating the cipher character for it.

In the example below we have introduced a shift modus to achieve a new permutation for each encryption step.

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

Image 3 *Changing Permutations*

We have created the ciphertext (***EQNVZ***) by mapping our key *k* from plaintext alphabet to each individual permutation alphabet, which is exactly the same as done in the OTP using modular arithmetic. The shift modus we used is not based on the Caesar cipher (*the shift modus in the Caesar cipher is fixed hence predictable*) as some comments we received suggested, but is based on the plaintext and the initial permutation alphabet, which will change every time we create a new cipher based on a new message.

The modus we used is very simple. We map our first character H which gives us the E and remove from the permutation up to this point the characters and place them at the end of our permutation. In doing so, we have created a new permutation, which will be used for the next encryption step. We repeat this step until our encryption is complete. In the system we suggest and developed the modus is different and ensures that none of the permutations becomes repetitive.

We have placed the example here to show that there is no fundamental difference between an operator choosing a key character or mapping between plaintext alphabet and permutation alphabet. What is different is the fact that an operator will always pick a key character, be that a mental exercise or a physical process, from a permutation that does not lead back to an initial permutation as shown here in the example in *Image* 3. But the fact remains that the OTP is a permutation cipher and the need for modular arithmetic arises because the receiving operator will not be able to reproduce the process that lead to each cipher character. It also creates a situation that requires us to send the cipher and the key to enable a recipient to decrypt the cipher we created. Using a changing permutation cipher as shown in our example above the need for the key transmission has been made obsolete because a recipient in the possession of the initial permutation can recreate the plaintext message by reversing the process we used to encrypt our message.

In theory there is the possibility that an adversary with enough computing power and time at hand will be able to break our cipher because the English alphabet contains only 26 characters and the resulting permutations are P26! (= 403,291,461,126,605,635,584,000,000). It might seem an incredibly large number, but with the computers of today (not taking into account the next generation of computers which will be quantum and memcomputers) it will not take too long to find our initial permutation. Our task in the next chapter now is to create an initial permutation that places an adversary into the same position as when he looks at a cipher using an OTP that has been created by an operator and all rules have been observed that grant perfect secrecy.

A different approach

*I*n this part we will apply three fundamental changes to the OTP moving away from the human operator and using random events to create our cipher by using random permutations for each single plaintext character to create our cipher. We will still comply with ***Kerkhoffs Principal*** (***everything about a system can be public knowledge only the key has to remain secret***) and the mathematics developed by ***Claude Shannon,*** which prove that **the OTP is the only theoretically perfectly secret form of encryption**, which **can not be broken** even using a brute-force attack.

In mathematics we are used to break *more complex* mathematical operation down into single step calculations to achieve a result:

(Example: $(a + b)^2 = (a + b)\ x\ (a + b) = a\ x\ a + a\ x\ b + b\ x\ a + b\ x\ b = \underline{a^2 + 2ab + b^2}$).

*U*sing an OTP we apply the same procedure by breaking a *complex* plaintext into single step operations when encrypting a plaintext (*see Image 1*). An operator picks for **each single plaintext character** a random *key* character **from a full set of the alphabet**. The problem one faces today is the definition that describes a One-Time Pad and its interpretation how it should operate and which seems to vary depending on the people that express their expertise on the subject. We will come to that subject in our conclusions and express our views on it.

*H*ere we will take the first step to apply the first change to our initial permutation sender and recipient have to exchange.

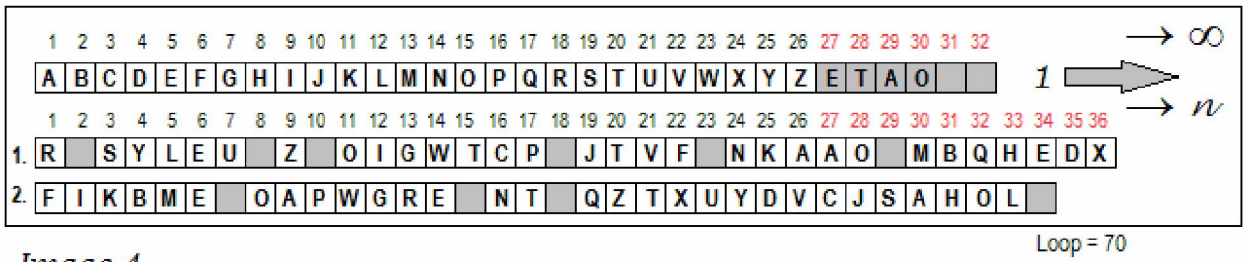


Image 4

*I*n the image above we have taken the English alphabet and extended it by six (6) characters and added four (4) of the most frequent letters used in this language to it and two (2) space characters. We could have added more characters or less and we could have picked different characters. For an adversary it has to be a string of unknown size, stretching from any rational number (n) to the theoretical possibility of being boundless (*infinite* - ∞). This string is now used to create **two** randomised strings to be exchanged between sender and recipient and these strings will be the secret they share.

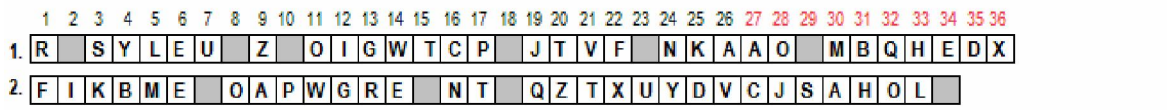
*O*ur second step is a change in the procedure how we record a cipher. We don't pick a random character as key character like we do in the original OTP or map them as in a permutation cipher; instead we record the difference between the plaintext characters as we encounter them on our two strings.

*T*he third step is a change in how we create new permutations when looping through our two strings and recording the differences between characters. Whenever we reach the end of our strings we will create two new randomised strings and after that continue recording the differences until encryption is complete.

Here is an example and how it operates. The message we encrypt is:

HELLO WE WILL MEET TOMORROW

First action we take is creating the two secret strings we will exchanged and create our initial permutations that will be used when we start our encryption. Here we used the value of 18 to achieve this. In the system that uses the ASCII (*extended set – hex numbers*) code we use a different modus to ensure that these strings are not repetitive.



We remove from both strings the first 18 characters and place them in *reversed* order at the back of our strings. These to strings are now the start points for our encryption. We look for the first character H and record the position; moving to the next character E and record the difference to the first character; move to the next character L and again record the difference to the previous character and we repeat this process until we have reached a point where the next character can not be found on the current string anymore. Let's now take a look at our example below and the first two strings. Here we could record the differences of the characters within the word HELLO and the following space character (string 2 – values 12 and 1). Now we have reached the end of the second string and the character W (*word WE*) isn't in the last 6 remaining characters. The system places the value of six into temporary storage and uses the last recorded value (*1 indicated by the red colour to the right of the strings*) as shift modus (*having only one character we have nothing to reverse here*) creating two new permutations. Now the temporary value of 6 is retrieved and we continue counting on newly created strings (*W is situated as position 22 + 6 = 28*).

This process is repeated until we have our complete plaintext encrypted.

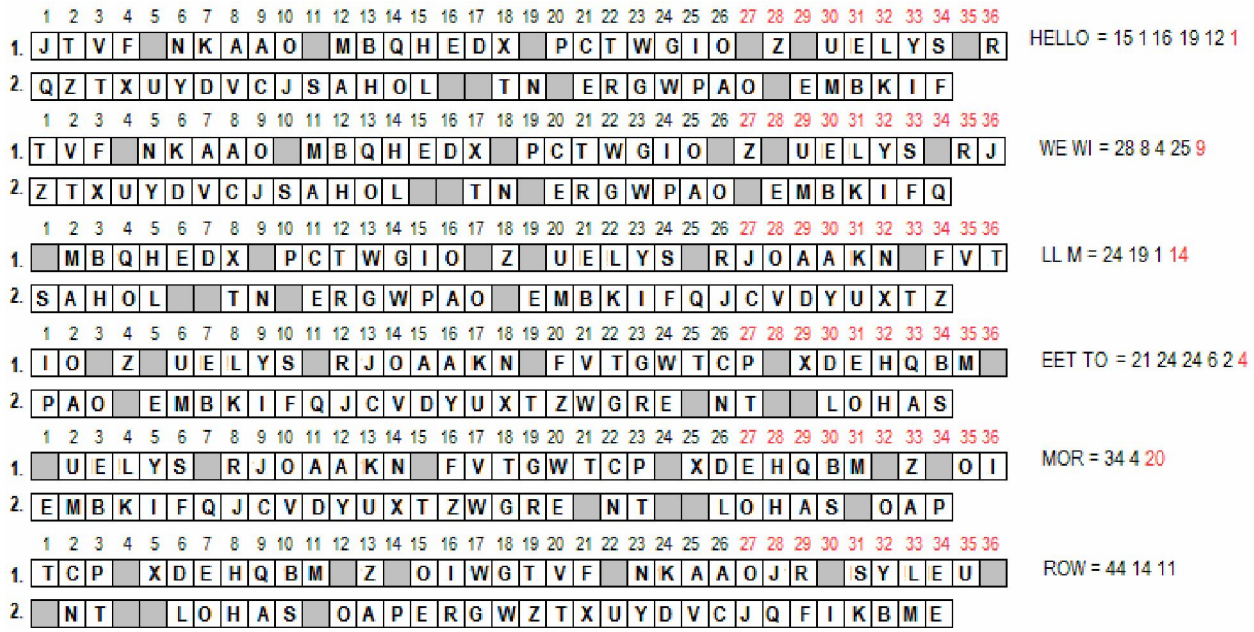


Image 5: The permutations created from initialisation and during encryption. Only two of them are in memory of the computer during encryption.

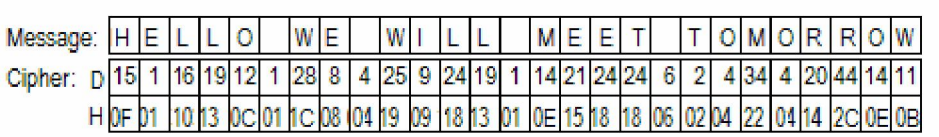


Image 6: Plaintext message, decimal numbers for each plaintext character and the hex code for each decimal number.

The result of our encryption can be seen above in *Image* 6. Let's consider what knowledge an adversary would have gained and how to attempt to break the cipher.
An adversary would know the length of our message and that it uses 27 characters. The adversary would also know everything about the system (*shift modus, two randomise permutations etc*). But what the adversary doesn't know is the length of our two string, which can be any rational number (*n*), or the characters we have inserted (*both strings can have different characters inserted; in our example on the previous page we used the same characters*).

Using a brute force attack wouldn't help because an adversary would have to rely on guessing about the length of our strings and the characters inserted since the numbers we recorded are differences between characters that all have a probability of 1/27 (due to the space character we added to our English alphabet). Using a 26 character alphabet would mean P26! possible permutations but here in the example an adversary is confronted with P*n*! permutations. An adversary could add all differences together and assume that we only use one string with the sum of all differences, still that would neglect the shifts we have in our example and the differences between characters are reset due to the shift modus. By going through all possible solutions an adversary would also end up with all possible plaintext options the cipher holds.

Would frequency analysis or any other tool that looks for pattern in languages help? Certainly not since frequencies of differences (numerical) between characters on randomised alphabetic strings which are changing frequently do not represent frequencies of characters or patterns that language holds.

Is the system we suggest still a one-time-pad?

Requirements as Claude Shannon put them:

1. The key must be truly random
2. The key must be as large as the plaintext (*or larger*)
3. The key can never be reused in whole or part

To maintain perfect secrecy when used as tool of communication Shannon adds a fourth point:

4. The key must be kept secret

1. Certainly we haven't violated any of the four rules above by changing from modular arithmetic to recording differences between characters. We maintain true randomness in as much as that expression can be defined correctly (*The key must be truly random*).

The problem we run into is the definition '*truly random*' when we try to explain what it means. There is no universally accepted definition for it and everybody seems to make one up when arguing their point for or against the OTP. So here is our contribution added to that confusion:

a) Unpredictability – non computability of bits in a sequence
b) Uniform distribution – bits in a sequence
c) Lack of patterns – in a sequence

meaning that **true randomness is non computability**.

2. Our key or keys will always match the length of our plaintext (*The key must be as large as the plaintext or larger*) because the modus used would make it impossible to do otherwise.

3. Here too we can say that our system doesn't permit to reuse the key in part or whole (*The key can never be reused in whole or part*) since our start permutations always will be non repetitive (ASCII extended set – hex numbers).

4. *The key must be kept secret* is another definition that can be argued about. If four or five people share a secret key or a modus that allows them to create and recreate this key and that state of secrecy can be maintained without any other person gaining access to it, we have secrecy. Still looking at the argument used by most of the opponents of the OTP, we will find that they point out, that the key after being used needs to be destroyed. Looking at the requirements set by Claude Shannon we can't find that this has to be the case. Nevertheless for Bruce Schneier and other opponents it has become a rule that is stipulated and a governing law when using an OTP.

Some mathematics

Readers that are not really have a taste for mathematics can skip this part because it is only of interest for people that want to know why the OTP in our view is a permutation cipher – not more not less – based on the rule that the length of each key is always 1 ($\ell = 1$).

It is not only the OTP that offers perfect secrecy but this also applies to shift ciphers (Caesar) and permutation ciphers when $\ell = 1$. When $\ell \geq 2$, then the rules that govern perfect secrecy will be violated.

Proof for $\ell = 1$
For each letter *m* and every $c \in C$ where $C = M = \{A,.. \text{ to } ,Z\}$ we have a unique key which *is* $k = c - m \bmod 26$. $Enc(k, m) = (m + k \bmod 26) = c$. This results in every *m, c* we have the $\underset{k \leftarrow K}{Prob}\ [Enc(k, m) = c] = 1/26$. This satisfies the rules of perfect secrecy.

Proof for $\ell \geq 2$
If *m1* = "*AB*" and *m2* = "*AD*" and *C* = "*EF*" then there is a key $k \in K$. If we encrypt it, it follows that $Enc(k,m1) = c$, to be precise $k = 1$. On the other hand for all $k \in K$ it concludes that $Enc(k,m2) = c$ and therefore $\underset{k \leftarrow K}{Prob}\ [Enc(K,m1) = c] = 1/26$, but $\underset{k \leftarrow K}{Prob}\ [Enc(K,m2) = c] = 0$ and with that we have violated the requirements requested by perfect secrecy.

Both proofs are base on a shift cipher but the same rules apply for a permutation cipher and the mathematics doesn't change because our key *k* is mapped from plaintext alphabet to ciphertext (*permutation*) alphabet. What is important is that each cipher character is mapped using a unique permutation which ensures that the resulting cipher character always has the probability of 1/26.

A system is perfectly secure or perfectly secret if knowing the cipher text gives no more information about the message that one would know without intercepting the encoded message at all, meaning, if $P(M = m) = P(M = m \mid C = c)$ for any $c \in C$ and any $m \in M$, regardless of what probability distribution is chosen on M. This is the same as stating that $P(C = c) = P(C = c \mid M = m)$ for any c and m. - Claude Shannon (1949)

In the OTP ℓ = (*always*) 1 since for each single plaintext character a random key character is selected from a full set of the alphabet (*in our examples the English alphabet*). **The choice has to be always 1/26 (*A to Z*) or it wouldn't be a One-Time Pad by definition and not providing us with a probability of 1/26 for each key character**; and that we know would not comply with the mathematics and the rules that govern the OTP (*or as Shannon puts it an ideal system*) which Claude Shannon described in his papers in 1949 (*Actually the papers were written in the mid 1940's but only declassified in 1949 – we might recall that during that time a war was fought on our planet, which made cryptology a sensitive issue. – We will write more on that subject and the situation in our days in our conclusions*).

What makes the OTP special and separates it from other encryption schemes is the fact that it needs a human operator and two separate transmissions between sender and recipient to claim the title of perfect secrecy. Apart from that it functions like a permutation cipher that changes after each encryption step the permutation to create a cipher.

Text, Audio, Video – any File Format

Up to now we have only looked at the possibility of text messages – without and with space characters – but we want to expand it and encrypt any existing file format and still have the protection the OTP offers.

In our paper we occasionally referred to the ASCII extended set and that we use it for encrypting and decrypting our messages. Actually what we are using are the character values 0 to F or 0 to 15 (16 numbers) expressed in decimal numbers. What we could have used too is UTF-8 to UTF-32 since they all use these 16 (0 to F) values to encode messages. What changes which each code page is the bytes we encode but the values when using hex numbers will always be 0 to F. ASCII uses 1 byte to represent characters and UTF-8 uses 1 to 4 bytes. ASCII in the extended set is represented by the h& values 00 to FF giving us 256 different values in hex code, which we decided is enough as an initial key string for our encryption system. If a value is above that 256 mark the system still will encrypt it using the values 0 to F (h&). For example the decimal number 257 would be represent in hex code using base 16 as h&0101 and as binary figure 0001 0000 0001.

For the encryption/decryption it is only important that the file is converted into random hex numbers regardless what the file contains or which code page (Chinese, Arabic, and Western European etc) was used to create this file and that after decryption the file is restored to its original state.

Using the same procedure (*but starting at the end of our string*) as outlined in our paper where users can change control characters (used for peripherals like printers, modems etc) and adding characters to the string (256 + *n*) will ensure that a brute force attack on the cipher will not only take a tremendous amount of time but also produce all possible decryption options our cipher might hold. The permutations for a string that contains 256 characters are P256! (8.578177753428426541190822716812 3e+506). Each character replaced within that string or added towards the string will change the amount of permutations an adversary would have to go through when applying a brute force attack.

Conclusions

Here a quote from an article written by Professor Keith Martin in the online publication *The Conversation*:

For one system, known as symmetric encryption, quantum computing doesn't pose much of a threat. To break symmetric encryption you need to work out which (of many) possible keys has been used, and trying all possible combinations would take an unimaginable amount of time. It turns out that a quantum computer can test all these keys out in one square root of the time it would take existing computers – in other words, slightly less time but not so dramatically that we need to worry.

But for another type of encryption system, known as asymmetric or public-key encryption, it doesn't look so good. Public-key systems are used for things like securing the data that comes through your web browser. They encrypt data using a key that is available to anyone but need another private key for decryption.

The private key is related to the public key, so to break the encryption you would need to perform a very difficult calculation that would give you the private key. This would take a conventional computer an impractical amount of

time. But when it comes to the two most common types of public-key encryption in use today, a quantum computer would be able to perform the calculations quickly enough to render them practically insecure.

*) **Keith Martin** is Professor at the Information Security Group, Royal Holloway in the UK. ***The Conversation*** Will superfast 'quantum' computers mean the end of unbreakable encryption?

We know from press releases and comments made by leading politicians that encryption software has become a touchy subject for the powers that try to explain to us how they see democracy and how it should function. Freedoms we took for granted like the right of privacy has been sacrificed to counter the threat of terrorism and organised crime. Mass surveillance has become the norm despite the fact that the terrorists and criminals are in the minority. Billions are spent to invent and upgrade tools that tighten the net that has been cast to ensure that no email, telephone call or other electronic communication goes unnoticed and gets recorded and filed away.

Since the revelations made by Edward Snowden we know further that suppliers of software and hardware haven't been honest with us when selling us products that promised security and secrecy for our thoughts and comments when transmitted via the Internet or using other forms of e-communications. We think it is time to take back a right that is fundamental for a democracy, which is the right of privacy and to express an opinion in communication exchanges with other people; without the fear that this exchange might be used against us should the political wind in the country change.

We would like to hear from people that have to come to the conclusion that the system we suggested in our paper to regain some privacy doesn't work and the reasons why; be that technical or mathematical objections. We would like with the permission of the commentators to publish their comments and our replies to get into a lively discussion on the subject.

For all the readers that have fought their way through the paper, a big thank you and we are looking forward to your comments.