The indecipherable cipher: The One-Time Pad

Daniel Alton

1 How did we get here?

As long as human communication has existed, there has been the need to make the distinction between the party you want to listen, and the party you do not want to listen. For millennia, that would just mean withholding information until you could speak in confidence, but as tribes grew into villages, villages grew into cities, and cities became subjects of empires, it became increasingly impossible to communicate by spoken word, especially as the number of things to communicate grew as the complexity of the nation did. A simple solution was designated messengers, the pinnacle of which, until the last few hundred years, was by horse, so long as you paid the courier well enough to ensure the privacy of your communications. This not always being possible, of course, fueled the invention of cryptography, and for centuries there was a fierce battle between the cryptographers working under powerful governments and interested parties to keep their secrets secret and the cryptanalysts working under equally powerful governments and parties interested in knowing those secrets.

The earliest and simplest encryption schemes are what would be called "monoalphabetic substitution ciphers", which is a big word for what simply means "one alphabet, but the letters are moved around a bit". Most people have heard of the Caesar cipher, which simply rotated the alphabet around itself to generate scrambled information. Vulnerable to frequency analysis, monoalphabetic substitution ciphers were replaced with polyalphabetic substitution ciphers, which for a while most thought were practically unbreakable, hence the name *Le Chiffre Indéchiffrables* (the indecipherable cipher). But it was decipherable, and so the battle continued. The most promising cipher was a kind of polyalphabetic (sort of) substitution cipher called the One-Time Pad, which encrypts each letter of information (although bits are more likely to be used when addressing modern cryptography; either is applicable for now) in such a unique and random way that, as long as the key is random, and as long as the message, the cipher is mathematically unbreakable. So, problem solved. Except not really. The One-Time Pad is certainly unbreakable if it is used correctly, but using it correctly is really not easy at all. To address the issues, we first need to understand the specific way in which the One Time Pad operates and encrypts/decrypts information. For the sake of brevity, and the fact that it is the atomic unit of information and what computers deal with when communication, we will treat a message as some string of bits. A string of bits is just some number base-2, which will come in handy later. The key thing here is that we can operate on the individual bits of information, and here's where the maths (or logic if you'd prefer) comes in.

2 The One-Time Pad

We begin with our message, our string of binary digits, m. m might look something like 10011000, but since we want to be operating on each bit of information individually, we'll index m, such that the first digit of m is $m_1 = 1$, and so on, for $m_2, m_3, \ldots, m_8$. We can't encrypt something without a key (quiet, hashing fans), so we will similarly define ourselves our key to be a string of binary digits $k = k_1 k_2 \ldots k_8$. To clarify, k is not the product of k at index 1 through 8, but the concatenation (just adding them in series); to distinguish multiplication from concatenation in the future, it should be clear that variables next to each other without explicit multiplication signs imply concatenation. k should be the same length as m for the One-Time Pad. Finally, the encrypted ciphertext, that is the unreadable message to be transmitted, should be defined in the same way as m, being a string of digits, but is not yet calculated.

The way that the one time pad encrypts information relies primarily off of a single logical operation, Exclusive OR. (XOR). If we XOR two binary numbers (XOR is commutative so the order does not matter), then the resulting bit will be 1 if the two inputs are different, and 0 if the inputs are the same. We can show this property in a truth table, where we represent XOR using the symbol $\oplus$.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

When we apply this to each pair of characters from m and k at a single index, we then find our appropriate ciphertext c, such that $m_i \oplus k_i = c_i$. Since we do this for each index in the length of the message, which is equal to the length of the key, then the resulting ciphertext has the same length. And that's all the One-Time Pad is! Digitally speaking, at least; when done on using alphabets longer than just 0 or 1 it is done slightly differently, but the principles are the same (In cryptography, the alphabet that is used in encryption is referred to as the *alphabet of definition* $\mathcal{A}$, a finite set. When dealing with binary, $\mathcal{A} = \{0, 1\}$). But what about decrypting? Well, let's think back to the original truth table, where $a \oplus b$ is shown, and lets imagine the result of this operation to be $a \oplus b = c$. What happens if we apply another layer of XORing? What happens if we find $c \oplus b$?

c	b	$c \oplus b$
0	0	0
1	1	0
1	0	1
0	1	1

Comparing a and $c \oplus b$, we find that they are equal! It turns out that XOR is the specific operation that negates itself when applied twice, meaning $(a \oplus b) \oplus b = a$. All we need to do to get a again is just XOR it with b again. Applying this to our message, where $c = m \oplus k$, we can simply rearrange to find that $m = c \oplus k$. The function to encrypt the message is the same function used to decrypt the message, making the One-Time Pad a relatively simple yet elegant method. How can such a simple function be not just strong, but unbreakable?

Both the fundamental advantages and disadvantages to the One-Time Pad can be traced to the fact that encryption occurs in an entirely isolated manner. The specific value at c_i depends only on the two values of m_i and k_i, and no other factors. Both m_i and m_i are single bits of information, one of two possible numbers; as shown in the truth table, there are only 4 possible combinations that they can be in. A cryptanalyst, hoping to discover the initial message, knows that for any value of c_i, there are two possible ways that they could be made; as shown in the first truth table, repeated below but renamed:

m_i	k_i	$m_i \oplus k_i = c_i$
0	0	0
0	1	1
1	0	1
1	1	0

If c_i is 0, you know that either m_i and k_i are both 0 or m_i and k_i are both 1. You do not know what the key is as an outside attacker, but you do know that $P(k_i = 0) = P(k_i = 1) = \frac{1}{2}$, since k and all of its digits is random. This is very important for knowing the plaintext message. Because $c_i = 0 \implies m_i = k_i$ and $c_i = 1 \implies m_i = \neg k_i$, regardless of what c_i actually is, it's a 50-50 chance for m_i to be 1 or 0 without knowing the exact value of k_i. As long as k_i remains random, that randomness is applied to the message to produce a ciphertext such that any possible message could produce any possible ciphertext given a key of the same length. Given a method of encryption, the set of all possible messages that can produce a ciphertext and the set of all possible keys are known as the *message space* $\mathcal{M}$ and *key space* $\mathcal{K}$. Because knowing an individual character of the ciphertext provides an equal probablity of the associated plaintext character being either 0 or 1, this scales up to the message as a whole - any ciphertext $c \in \mathcal{C}$ (The set of all possible ciphertexts for a given length) could be produced by any plaintext $m \in \mathcal{M}$, and vice-versa. The ciphertext is the product of equal parts the key and the plaintext; without knowing either, every possible input is equally likely. This is why the One-Time Pad is so secure - its cryptographic integrity doesn't depend on the complexity of the operations, but simply the logical conclusion that in the operation $m \oplus k = c$, the set of possible values for both m and k are equal in length to the total possible number of binary strings of that length. The One-Time Pad is a truly fantastic example of how simple can sometimes be better.

3 Mathematical integrity $\neq$ practical integrity

Mathematically, the One-Time Pad really is unbreakable when used properly, as proven by Claude Shannon in the 1940s. However, it's that "when used properly" that makes the One-Time Pad unsuited to a lot of practical applications. There are 4 requirements that must be followed in an encryption scheme in order for the One-Time Pad to remain unbreakable:

1. The key must be kept secret.
2. The key must be at least as long as the message to be encrypted.
3. The key must be entirely and truly random.
4. The key must be used only once (once for encryption, once for decryption).

Of course, the idea that the key must be kept secret is obvious, and applicable to every other keyed encryption method. The other three requirements, however, are increasingly difficult to approach. To begin with, the second requirement at first seems trivial. Of course, a method of encryption that works uniquely per bit must have as many unique bits in the key as bits it needs to encode. But in comparison to standard symmetric encryption schemes the effect is quite significant - for a small message, it is unimportant if the key is as long as the message. But if you are encrypting a vast sum of data, perhaps gigabytes of sensitive information, you need an equal quantity of data storage for the key - a 10 gigabyte image, encrypted via One-Time Pad, would have to use a 10 gigabyte key, which would have to be securely distributed. In comparison, a block cipher doesn't care how long the message is, and often keys might be 128 bits. Such a vital aspect of cryptography is the ability to encrypt a message using less information than the message itself carries, for ease of usage and key distribution. While all symmetric cipher methods suffer from the key distribution problem, the One-Time Pad physically doubles the amount of information that must be transferred compared to the message, and necessitates a key that is generally orders of magnitude larger than block ciphers, which can operate similarly efficiently. The potential for the key to be incredibly long factors in massively to the next point: The key must be entirely and truly random. When we are generally dealing with digital data, how do you generate a random number? When you need a key gigabytes in length, how do you generate billions of random bits? Computers are deterministic, for the most part, and to generate true random numbers is very difficult. Often, the best computers can do are pseudorandom number generators, which use convoluted (but still deterministic) algorithms, such as the Mersenne twister, but given enough information these are not *truly* random. This issue is only exacerbated by the final requirement - the key cannot even be used. For every bit of data being sent, another bit must be generated randomly for the key, and no two messages can use the same key. But why is this? The first three requirements are self-evident from the mode of operation of the One-Time Pad, but the latter might need some more explanation.

Let's say we have again a key k, and two messages m_1 and m_2. The ciphertexts can be found as

$$c_1 = m_1 \oplus k$$

$$c_2 = m_2 \oplus k$$

Here, m_1 is message 1, not the 1st digit, and likewise with m_2, c_1, c_2. If an attacker finds both c_1 and c_2, and is aware that they were both encrypted using the same key, k (For the sake of simplicity this is assuming that both m_1 and m_2 are of equal length), then they know that

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k)$$

XOR is a commutative operator, therefore

$$(a \oplus c) \oplus (b \oplus c) = a \oplus b \oplus (c \oplus c)$$

Anything XORed with itself is necessarily 0, and anything XORed with 0 is necessarily itself. It follows from this that

$$a \oplus b \oplus (c \oplus c) = a \oplus b \oplus 0 = a \oplus b$$

$$(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

The XOR of both ciphertexts equals the XOR of both plaintexts. The attacker can use a known-plaintext attack to reverse one of the plaintexts if they know the other. But most importantly, $m_1 \oplus m_2$ has no aspect of the key's randomness - even the XOR of two human texts can be vulnerable to frequency analysis. Frequency analysis still isn't perfect, but as the message grows in length it becomes more effective, and increasingly verifiable.

In conclusion, it is these limitations that mean that the One-Time Pad, though truly unbreakable *mathematically*, suffers from practical issues and requirements that may not always make it very useful. It suffers from the same key-distribution problem as other symmetric encryption methods, requires a vast quantity of random data to be generated and transmitted, and if misapplied can lead to disastorous lapses in security. The One-Time Pad is mathematically elegant, and cryptographically impregnable, but practically brittle and unforgiving. Alone, it is simply not sufficient to ensure reasonably efficient privacy, and a prime example of how mathematical perfection is only as perfect as our ability to apply it.