Introduction to Cryptography

Build your first own encryption algorithm

Oliver Snellman*

November 2023

Introduction

Encryption is a process that transforms a message into a coded message. The idea is to make it easy for a friend to recover the original message, but very hard for anyone else to do the same. The coded message can be transformed back to the original message only with a private key, in a process called decryption.

If you intercepted a message and knew how it was encrypted, could you decrypt that message? Surprisingly, no! Actually, the functioning of the popular Secure Hash Algorithm (SHA-256), used for example in many blockchain applications, is common knowledge. How can you not, then, deduce what the input was that went into the algorithm, based on the observable output and detailed understanding of what the algorithm does to the input?

Reversing modern encryption algorithms in this way is not possible, because they introduce pseudo-randomness to break the one-to-one mapping between the message and the coded message. Therefore, knowing the rules of the algorithm doesn't help, if the pseudo-randomness can't be negated, which is only possible with something called the "private key".

Let's build a simple encryption system from scratch to gain intuition on how securing messages happens nowadays.

Mapping

Consider a function that maps the 26 letters in the English alphabet into the first 26 natural numbers, creating a lookup table. More technically, this function is an injection from the domain (message space) to the range (coded message space).

$$\begin{tikzcd}
\textbf{Input} & \text{A} \arrow[dd] & \text{B} \arrow[dd] & \text{C} \arrow[dd] & \cdots & \text{Z} \arrow[dd] \\
\textbf{Mapping} & & & & & \\
\textbf{Output} & 1 & 2 & 3 & \cdots & 26
\end{tikzcd}$$

Figure 1: Simple encryption with one-to-one mapping.

Using this rule, any text can be encrypted into an unreadable sequence of numbers.

*University of Helsinki, visiting the University of Pennsylvania, Email: oliver.snellman@gmail.com

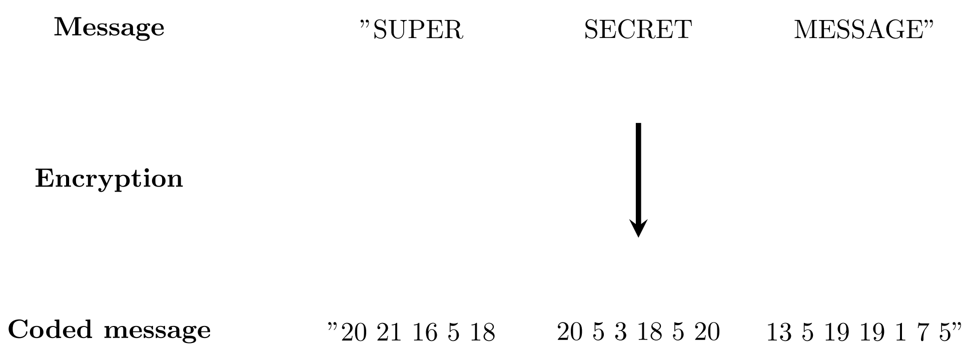

Figure 2: Transforming a sentence into numbers with a one-to-one mapping.

If there are many coded messages encrypted with this scheme, it is possible to observe patterns, use probabilities of how commonly different words incur in language, and to deduce the logic behind the encryption. Technically this amount to finding the inverse of the injective encryption function. For example, it might be easy to deduce the mapping for short words like "I"="9" or "AND"="1 14 4" and use those to deduce the mapping for the rest of the letters. If someone does figure out the lookup table in Figure 1 then it is easy to decrypt all messages which were encrypted using the same method.

Adding pseudo-randomness

Modern encryption algorithms break the one-to-one mapping in a way that can't be easily reversed. Let's start by looking at two examples of encryption, which break the one-to-one mapping.

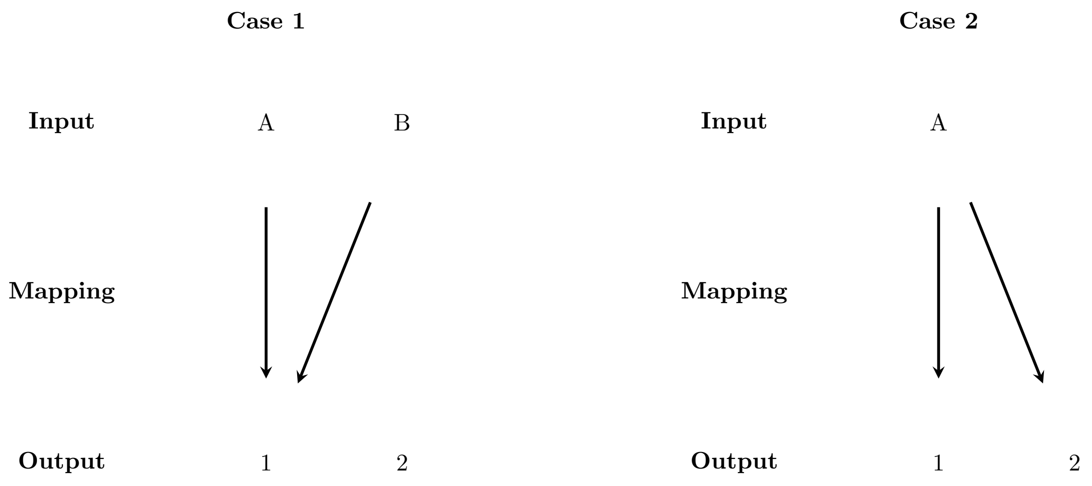

Figure 3: The encryption in Case 1 allows multiple inputs to map into the same output (non-injective). The encryption in Case 2 allows an input to map into different outputs at different times (correspondence). An encryption can allow both to happen at the same time.

One way to break the one-to-one mapping, like in the Figure 3, is by introducing pseudo-randomness. **Pseudo-random** number generator produces numbers which look completely random, but the same exact numbers can be re-produced with a **Private key** (seed).

Let's say Harry wants to encrypt the word "MOON" and send it to Remus. Consider the same mapping as before in Figure 1, but now we add a second step: To each of the numbers resulting from the mapping we add randomly either -1, 0, or +1, each with 1/3 probability.

$$\begin{tikzcd}
\textbf{Message} \arrow[d, phantom, "\textbf{Mapping}"] & \mathrm{M} \arrow[d] & \mathrm{O} \arrow[d] & \mathrm{O} \arrow[d] & \mathrm{N} \arrow[d] \\
\textbf{Stage 1} \arrow[d, phantom, "\textbf{Add randomness}"] & 13 \arrow[d, "+1"] & 15 \arrow[d, "0"] & 15 \arrow[d, "-1"] & 14 \arrow[d, "0"] \\
\textbf{Coded message} & 14 & 15 & 14 & 14
\end{tikzcd}$$

Figure 4: Encryption with pseudo-randomness, controlled with a private Key.

With this encryption algorithm and with the particular random numbers (+1, 0, -1, 0), the message "MOON" transforms to the coded message "14 15 14 14". Now there are three different instances of "14" in the coded message, each corresponding to a different letter in the original message, demonstrating the Case 1 from Figure 3. The letter "O" in the original message is also now represented by two different symbols in the coded message, "15" and "14", demonstrating the Case 2.

If we were to augment the original message by writing the word twice "MOON MOON", the second coded word "MOON" could look different from the first one after the encryption. Therefore, in absence of one-to-one mapping, it is not possible to break the code and construct a lookup table to cipher the future messages.

But how can anyone, even the correct recipient, then, recover the original message? The answer lies in the private key. It is not possible for computers to come up with completely random numbers, but instead they generate the numbers deterministically using some complex function, which is very sensitive to its input values[1]. This function is supplied with some predetermined list of input values, which create the random looking outcomes. By setting a "seed" we specify that a particular list of input values is to be used in the pseudorandom number generation. With the same seed, the same numbers will come out every time. Hence, the random numbers only appear to be random, but they can be recreated with the seed. The seed is the private key in this example.

If two people have agreed to always use the number "2039" as the private key, alongside with a specific function with which to create the random looking numbers - then they can send each other encrypted messages in public, which only they can decrypt. The following R-code demonstrates the issue.

[1]Just in case it's unclear, these input values are conceptually different from the inputs (letters) to the encryption algorithm.

```
No seed                                      Seed

                                             set.seed(2039)
sample(c(0,-1,1), 4, replace=TRUE)           sample(c(0,-1,1), 4, replace=TRUE)
[1] 0 0 -1 1                                 [1] 1 0 -1 0

                                             set.seed(2039)
sample(c(0,-1,1), 4, replace=TRUE)           sample(c(0,-1,1), 4, replace=TRUE)
[1] 1 1 -1 0                                 [1] 1 0 -1 0
```

The recipient of the encrypted message "14 15 14 14" can now generate the same four random numbers (+1, 0, -1, 0), deduct them from the encrypted message (as we are now going in the opposite direction), and then apply the simple mapping from Figure 1 in reverse. This procedure yields the original message "MOON".

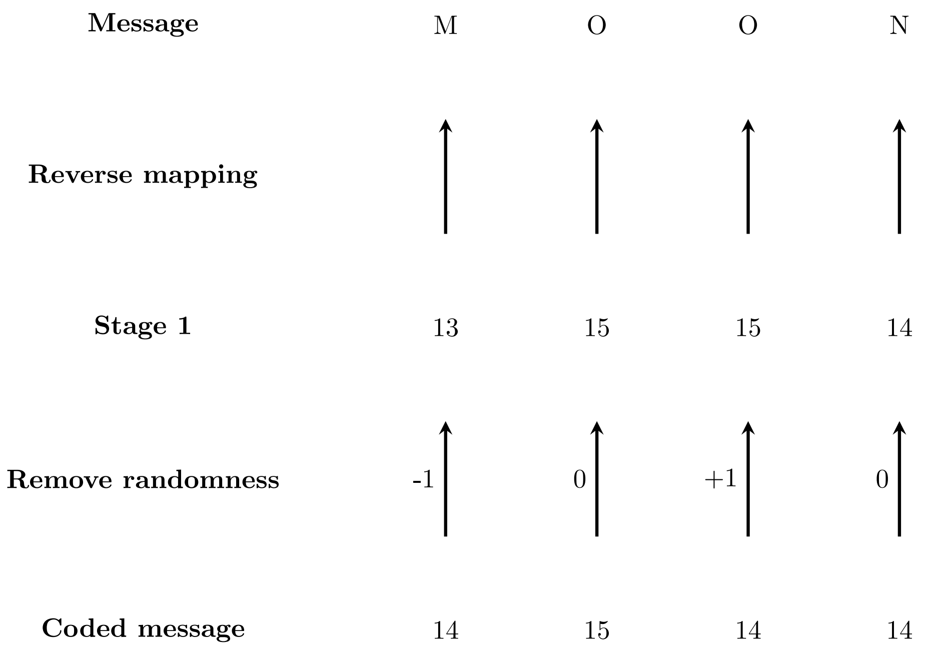

Figure 5: Decryption using the private key "2039".

Final step

The encryption typically maps input characters into output characters, instead of numbers. The problem with numbers is the need to use more than one digit to represent some inputs.

We add the same initial mapping but in reverse as the final step of encryption, to get the coded message in characters. The decryption happens similarly as in Figure 5, but with the additional initial step of first reverse mapping the coded message to numbers.

$$\begin{tikzcd}
\textbf{Message} & M \arrow[dd] & O \arrow[dd] & O \arrow[dd] & N \arrow[dd] \\
\textbf{Mapping} & & & & \\
\textbf{Stage 1} & 13 \arrow[dd, "+1"] & 15 \arrow[dd, "0"] & 15 \arrow[dd, "-1"] & 14 \arrow[dd, "0"] \\
\textbf{Add randomness} & & & & \\
\textbf{Stage 2} & 14 \arrow[dd] & 15 \arrow[dd] & 14 \arrow[dd] & 14 \arrow[dd] \\
\textbf{Reverse mapping} & & & & \\
\textbf{Coded message} & N & O & N & N
\end{tikzcd}$$

Figure 6: The full encryption algorithm.

Further

In our encryption algorithm there are three main components: First, mapping from letters to numbers, then adding pseudo-randomness to those numbers and finally a second mapping back from numbers to letters. So far we have specified quite simple solutions to all of these steps.

The mapping between characters into numbers can be improved by adopting for example the Unicode Transformation Format (UTF-8). In R the function utf8ToInt() maps letters and special characters (and even numbers) to numbers. Also the space " " between words in the message should be encrypted and the empty space should be added to the output options, to break the sentence structure in encryption. The inverse of this function is intToUtf8(), although the second mapping does not necessarily need to be the inverse of the first mapping.

The pseudo-randomness can also be added in much more versatile ways. For example, we could allow any input to be mapped into any of the output options, instead of just the adjacent ones. There can be many private keys and multiple stages of adding more mappings and randomness.

Hashing algorithm is another popular cryptographic tool. Unlike encryption algorithms, hashing algorithm is not meant to be reversed even by the friendly recipient. Suppose there is a publicly observed document and a sequence of numbers called the public key. When a hashing algorithm is used on the document with the private key, it outputs the public key. Only someone who knows the private key can use the hashing algorithm to re-create the public key, thereby having the power to verify the document or to claim ownership of it. Probably the best known example of this is the Secure Hash Algorithm (SHA-256), developed by the NS. It is used in blockchains behind many cryptocurrencies. Without knowing the private key, the best way is to guess a random private key and to check whether the SHA-256 algorithm produces the public key with it. The modern encryption algorithms have been designed in such a way that it is infeasible even with the best computers to obtain the private key this way.