2021-03-04 theory

March 2021

1.4.1 The operation $\boxtimes$

For a given $a, b \in 0, 1^5$ the operation $\boxtimes$ is the multiplication in base 2 disregarding bits of positions $> n$ (in the example below $n = 5$, just for teaching purposes). For example, if a $=$ [01010] and b $=$ [10011] then we can compute a $\boxtimes$ b by hand as:

$$\begin{array}{ccccccccc}
 & & & & 0 & 1 & 0 & 1 & 0 \\
 & & & \boxtimes & 1 & 0 & 0 & 1 & 1 \\
\hline
 & & & & 0 & 1 & 0 & 1 & 0 \\
 & & & 0 & 1 & 0 & 1 & 0 & \\
0 & 1 & 0 & 1 & 0 & & & & \\
\hline
 & & & & 1 & 1 & 1 & 1 & 0
\end{array}$$

Actually, if a,b are regarded as integers in base 2 , i.e. a $=$ 10,b $=$ 19, then the performed operation is: a $\boxtimes$ b $=$ (a $\cdot$ b)$(mod 2^n)$. In the current example:

$$10 \boxtimes 19 = 30$$

because $10 \cdot 19 = 190 = 30 (mod 32)$ (in python (10*19)%32 gives 30).

1.4.2 A baby example of asymmetric cipher

Bob's pair (sk, pk): the public key is pk $=$ 19 and the secret key sk is the x such that $19 \boxtimes x = 1$. Here is how Alice use Bob's public key to encipher $P \in 0, 1^5$:

$$\begin{tikzcd}
\textbf{Alice} & & \textbf{Bob} \\
C = \mathsf{Enc}_{pk}(P) = P \boxtimes 19 \arrow[rrd, "c"] & & \\
 & & P = \mathsf{Dec}_{k}(C) = C \boxtimes \mathsf{sk}
\end{tikzcd}$$

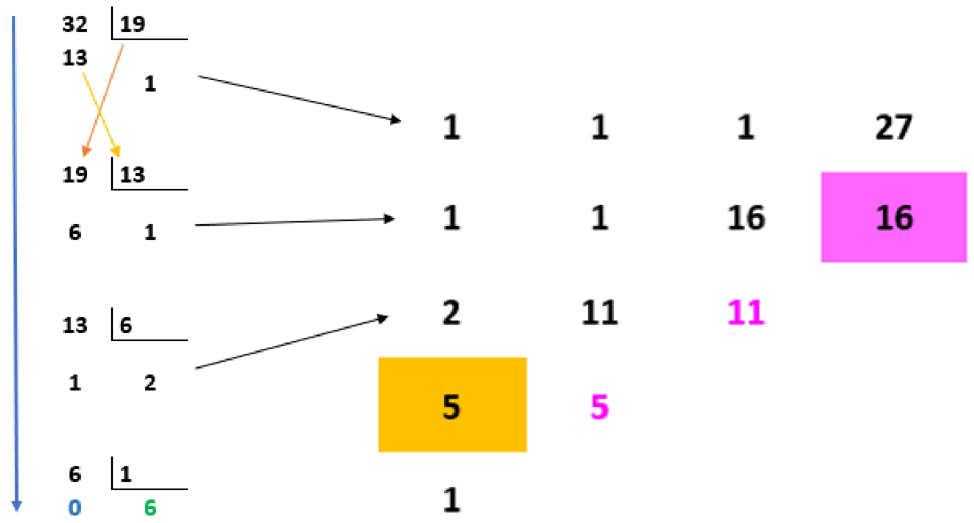
Is it possible to find Bob's secret key sk in the case above?
There are two possible solutions to this exercise:

- By means of a brute force loop on python
- Using the Kuttaka algorithm

The Kuttaka algorithm
Hypotesis: the pair (sk, pk) is calculated by: pk ⊠ sk = 1, so that the sk is the inverse of the pk.
The Kuttaka algorithm allow to find the secret key (sk) with calculations made by hand, starting from a public key (pk), by computing the inverse of numbers.
Example of Kuttaka algorithm implementation:
Bob's pk is [10011] = 19. The used strategy is:

$$19\cdot \equiv 1(mod32) \leftrightarrow 19 \boxtimes x \equiv 1(mod32)$$

Then, multiple divisions are needed, starting from 2^n/pk, until a remainder with value 0 is reached:



The previous table is created with these rules:

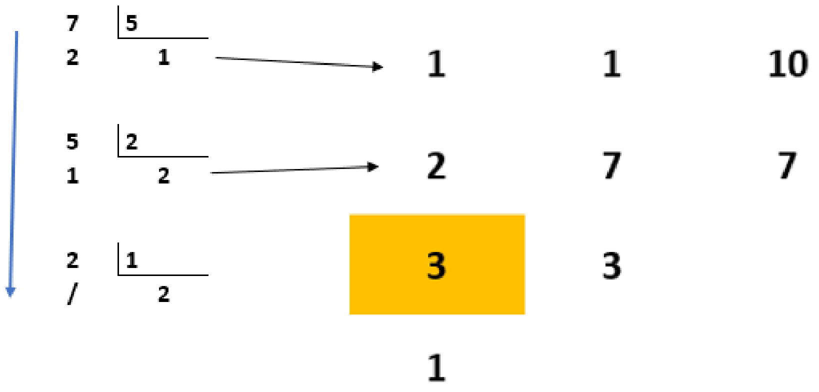
1. The first column has M = X + 1 cells, where X is the number of division performed;
2. The first values of the first column are copied from the results of the divisions;

3. The last values is always 1;
4. The second-to-last value of the first column is computed as follows:
   - if M is even, then the value is computed by subtracting 1 to the result of the last division
   - if M is odd, then the value is computed by adding 1 to the result of the last division
5. The second-to-last value of each other column is computed as the sum of the second-to-last values of previous columns. In the example: $16 = 11 \cdot 5$.
6. Other numbers are just copied to the next column
7. The final result is contained in the top-right corner of the table

In order to verify the final result it is possible to use python:

$$19^-1 \equiv 27(mod 32)$$

Another example of Kuttaka algorithm: $\frac{3}{5}$ in $\mathbb{Z}_7$:



1.5 A baby Galois ⊗-cipher

Following Evariste Galois we introduce a new multiplication $\otimes$ between strings $a, b \in 0, 1^5$:

$$a \otimes b = c$$

Be careful to not confuse $\otimes$ with a XOR ($\oplus$).

Let's take a Galois modular function G(x) and two polynomials a(x) and b(x), coming from $a = [a_4 a_3 a_2 a_1 a_0]$ and $b = [b_4 b_3 b_2 b_1 b_0]$, respectively:

$$a(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
$$b(x) = b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

The Galois multiplication consist in compute $(a(x) \times b(x)) mod G(x)$, with a remainder $c(x)$.
As an explicit example let us compute [01010] $\otimes$ [10011]. First of all we compute

$$(0x^4 + 1x^3 + 0x^2 + 1x + 0) \times (1x^4 + 0x^3 + 0x^2 + 1x + 1)\,.$$

Namely:

$$\begin{array}{ccccccccc}
 & & & & 0x^4 & 1x^3 & 0x^2 & 1x & 0 \\
 & & & \times & 1x^4 & 0x^3 & 0x^2 & 1x & 1 \\
\hline
 & & & & 0x^4 & 1x^3 & 0x^2 & 1x & 0 \\
 & & & 0x^5 & 1x^4 & 0x^3 & 1x^2 & 0x & \\
0x^8 & 1x^7 & 0x^6 & 1x^5 & 0x^4 & & & & \\
\hline
0x^8 & 1x^7 & 0x^6 & 1x^5 & 1x^4 & 1x^3 & 1x^2 & 1x & 0
\end{array}$$

and we get $x^7 + x^5 + x^4 + x^3 + x^2 + x$. Now its remainder when divided by $x^5 + x^2 + 1$.

$$\begin{array}{llll|l}
x^7 + x^5 & + x^4 + x^3 & + x^2 + x & & x^5 + x^2 + 1 \\
x^7 & x^4 & x^2 & & \hline x^2 + 1 \\
\cline{1-4}
x^5 & x^3 & x & & \\
x^5 & x^2 & 1 & & \\
\cline{1-4}
0 & \underbrace{x^3 + x^2 + x + 1} & & &
\end{array}$$

So

$$[01010] \otimes [10011] = [01111]\,, \text{ or } \; 10 \otimes 19 = 15$$

In the end, the result is: [01111] $(modG(x))$.

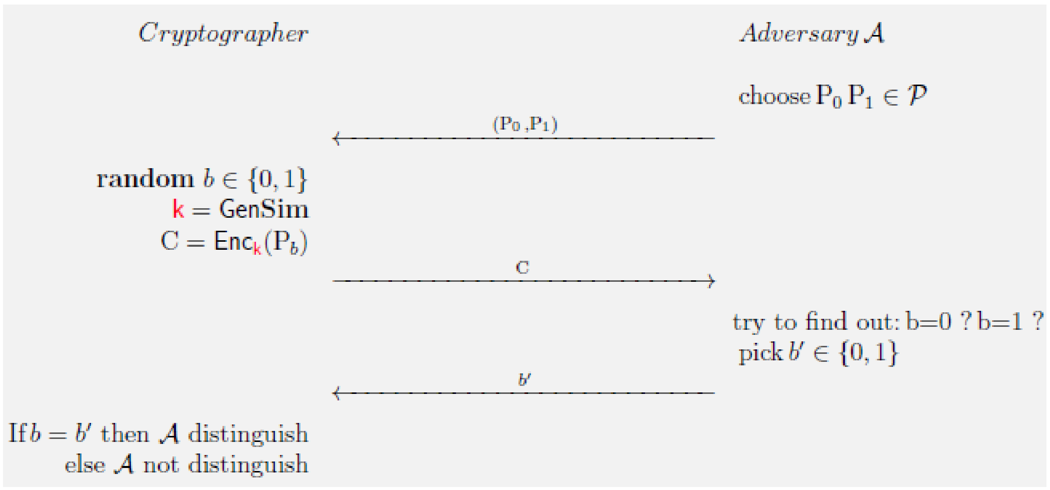
Bob's pair (sk, pk): the public key is pk = 19 and the secret key sk is the x such that $19 \otimes x = 1$.
Here is how Alice use Bob's public key to encipher $\mathrm{P} \in \{0,1\}^5$:

$$\begin{array}{lcl}
\textbf{Alice} & & \textbf{Bob} \\
\mathrm{C} = \mathrm{Enc}_{\mathrm{pk}}(\mathrm{P}) = \mathrm{P} \otimes 19 & & \\
 & \xrightarrow{\quad\quad \mathrm{C} \quad\quad} & \\
 & & \mathrm{P} = \mathrm{Dec}_{\mathrm{k}}(\mathrm{C}) = \mathrm{C} \otimes \mathrm{sk}
\end{array}$$

1.6 Symmetric Cryptography

1.6.1 IND Indistinguishability experiment

Ciphertext indistinguishability is a property of many encryption schemes. Intuitively, if a cryptosystem possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. The property of indistinguishability under chosen plaintext attack is considered a basic requirement for most provably secure public key cryptosystems, though some schemes also provide indistinguishability under chosen ciphertext attack and adaptive chosen ciphertext attack.
IND is a theoretical experiment used to evaluate indistinguishability of a cryptosystem. In this experiment there is a cryptoanalyst (adversary) that sends us two different plaintexts. We, acting as a cryptographer, have to randomly choose one of the two using a *coin mechanism* (like heads and tails game) and encrypt it. If the cryptoanalyst will be unable to distinguish our random choice (head or tail), then the cryptosystem described by this specific experiment is considered secure regarding the indistinguishability.
The experiment is repeated several times and a probability of the chance to distinguish the random choice is calculated (probability $p \in [0, 1]$).


Cryptographer
Adversary $\mathcal{A}$
choose $P_0\, P_1 \in \mathcal{P}$
(P_0, P_1)
random $b \in \{0,1\}$
$k = \mathsf{GenSim}$
$C = \mathsf{Enc}_k(P_b)$
C
try to find out: b=0 ? b=1 ?
pick $b' \in \{0,1\}$
b'
If $b = b'$ then $\mathcal{A}$ distinguish
else $\mathcal{A}$ not distinguish

The symmetric cryptosystem $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ is defined as **IND-secure** if and only if no adversary $\mathscr{A}$ can distinguish with probability better than $1/2$.

1.7 CPA-IND experiment

Another type of experiment applicable to symmetric cryptosystems is the CPA-IND (Chosen Plaintext Attack INDistinguishability).

This experiment is similar to the previous one and it is made up different steps:

1. A n-bit key k is generated by running Gen function
2. The adversary (cryptoanalyst), connected to the server (cryptographer) having access to $Enc_k(\cdot)$, chooses two plaintexts P_0 and P_1 of the same length
3. A random bit $b = 0, 1$ is chosen and then a ciphertext C is computed and sent to the adversary.
4. Adversary, who has access to $Enc_k(\cdot)$, generates b'
5. If $b' = b$ then adversary has succeded to distinguish, otherwise not.

The symmetric cryptosystem $\Pi =$ (Gen, Enc, Dec) is defined as **CPA-IND-secure** (or just CPA-secure) if and only if no adversary $\mathscr{A}$ can distinguish with probability better than $1/2$.

Which are the differences from previous case?
Inside the server there is an $Enc_k(\cdot)$ program running that encrypt according to the symmetric key k and encryption is <u>not</u> deterministic! Before we had that, given a plaintext P, the correspondant ciphertext is always the same. Instead, in this case result could be C now, but then, if we re-use the same encryption function on the same plaintext, it could generate C', C", etc. different from C. This property is due to the fact that encryption is based also on a random sequence of bits:

$$\begin{gathered}
\text{C} = Enc_k(random_{bits}||P_b) \\
\text{C'} = Enc_k(random_{bit_s'}||P_b) \\
\text{C"} = Enc_k(random_{bit_s''}||P_b) \\
\text{...and so on...}
\end{gathered}$$

Note: A <u>good</u> client will knows the length of the random sequence and will throw away the right number of bits in order to keep only the enciphered plaintext. So, for a good client, doesn't matter if it receives C, C', C" or others.