# CS254 (Spring 2018): Lab Assignment 3

January 23- January 30, 2018

In this lab assignment, you are required to implement an encrypter module and a decrypter module for our own simple, custom encryption algorithm. The encryption algorithm takes as input a 32-bit plaintext word $P = p_{31}p_{30}\dots p_0$ and a 32-bit key $K = k_{31}k_{30}\dots k_0$, and outputs a 32-bit ciphertext word $C = c_{31}c_{30}\dots c_0$. The decryption algorithm similarly takes as input a 32-bit ciphertext word $C = c_{31}c_{30}\dots c_0$ and a 32-bit key $K = k_{31}k_{30}\dots k_0$, and outputs a 32-bit plaintext word $P = p_{31}p_{30}\dots p_0$.

The encryption algorithm works as follows:

1. Let $N_1$ be the count of 1's in $K$ and let $T$ be a 4-bit vector.
2. $C \leftarrow P$.
3. $T_3 \leftarrow k_{31} \oplus k_{27} \oplus k_{23} \oplus k_{19} \oplus k_{15} \oplus k_{11} \oplus k_7 \oplus k_3$.
4. $T_2 \leftarrow k_{30} \oplus k_{26} \oplus k_{22} \oplus k_{18} \oplus k_{14} \oplus k_{10} \oplus k_6 \oplus k_2$.
5. $T_1 \leftarrow k_{29} \oplus k_{25} \oplus k_{21} \oplus k_{17} \oplus k_{13} \oplus k_9 \oplus k_5 \oplus k_1$.
6. $T_0 \leftarrow k_{28} \oplus k_{24} \oplus k_{20} \oplus k_{16} \oplus k_{12} \oplus k_8 \oplus k_4 \oplus k_0$.
7. **for** i = 0 **to** $N_1 - 1$ **in steps of** 1
   (a) $C \leftarrow C \bigoplus TTTTTTTT$ (component-wise xor of $P$ and $T$ cncatenated with itself 8 times).
   (b) $T \leftarrow T + 1$ (32-bit addition ignoring carry generated at the most significant bit)
8. Output $C$.

Similarly, the decryption algorithm works as follows:

1. Let $N_0$ be the count of 0's in $K$ and let $T$ be a 4-bit vector.
2. $P \leftarrow C$.
3. $T_3 \leftarrow k_{31} \oplus k_{27} \oplus k_{23} \oplus k_{19} \oplus k_{15} \oplus k_{11} \oplus k_7 \oplus k_3$.
4. $T_2 \leftarrow k_{30} \oplus k_{26} \oplus k_{22} \oplus k_{18} \oplus k_{14} \oplus k_{10} \oplus k_6 \oplus k_2$.
5. $T_1 \leftarrow k_{29} \oplus k_{25} \oplus k_{21} \oplus k_{17} \oplus k_{13} \oplus k_9 \oplus k_5 \oplus k_1$.
6. $T_0 \leftarrow k_{28} \oplus k_{24} \oplus k_{20} \oplus k_{16} \oplus k_{12} \oplus k_8 \oplus k_4 \oplus k_0$.
7. $T \leftarrow T + 15$.
8. **for** i = 0 **to** $N_0 - 1$ **in steps of** 1

(a) $P \leftarrow P \bigoplus TTTTTTTT$ (component-wise xor of $P$ and $T$ concatenated with itself 8 times).

(b) $T \leftarrow T + 15$ (32-bit addition ignoring carry generated at the most significant bit)

9. Output $P$.

Implement an encrypter and a decrypter module in VHDL, design an adequate testbench for it, and show your design and simulation waveforms.

**Important Note:** You **must** use a `process` statement with a *clock* on its sensitivity list and a conditional statement of the form `if (clock'event and clock = '1')` in the body of the `process` to implement the sequencing of statements from one loop iteration to the next in both the encrypter and decrypter modules. See the template file provided to see how to write such a `process` statement. When simulating the design, the body of a `process` statement gets executed whenever there is a change in any of the signals in its sensitivity list. The conditional statement `if (clock'event and clock = '1')` executes only when *clock* has a transition (i.e. the `clock'event` condition evaluates to true), and the new value of *clock* is 1. This effectively means that *clock* had a positive (0 to 1) transition. Thus, all `signal`s assigned with the `<=` operator and within the scope of the `if (clock'event and clock = '1')` statement can change only after the clock has a positive edge. Effectively, all such `signal`s become outputs of D flip-flops that are triggered by the positive edge of the *clock*. In fact, all assignments with `<=` in the body of such a `process` statement and in the scope of `if (clock'event and clock = '1')` happen concurrently at the outputs of the corresponding D flip-flops immediately **after** the positive edge of the clock arrives. The values of each of these `signal`s stay unchanged between two consecutive positive edges of the clock. Note that this is consistent with the behaviour of a positive edge-triggered D flip-flop, in which the input gets copied to the output only when a positive edge of the clock arrives.

A sample VHDL file with a template of the design is provided for your convenience.