Cryptography & Blockchain – 5IF, INSA Lyon

DES LAB REPORT

Participants: Lancelot TARIOT CAMILLE, Aymerick YZIDEE, Ahmed MANSOUR

Warning:

You will be able to find implementations in the Moodle submission box along with this report, but not everything could be submitted there (limited to 5 files).

For the full code and to prove that we followed every step of the lab, kindly refer to our public repository:

https://github.com/LancelotTC/Lab-DES

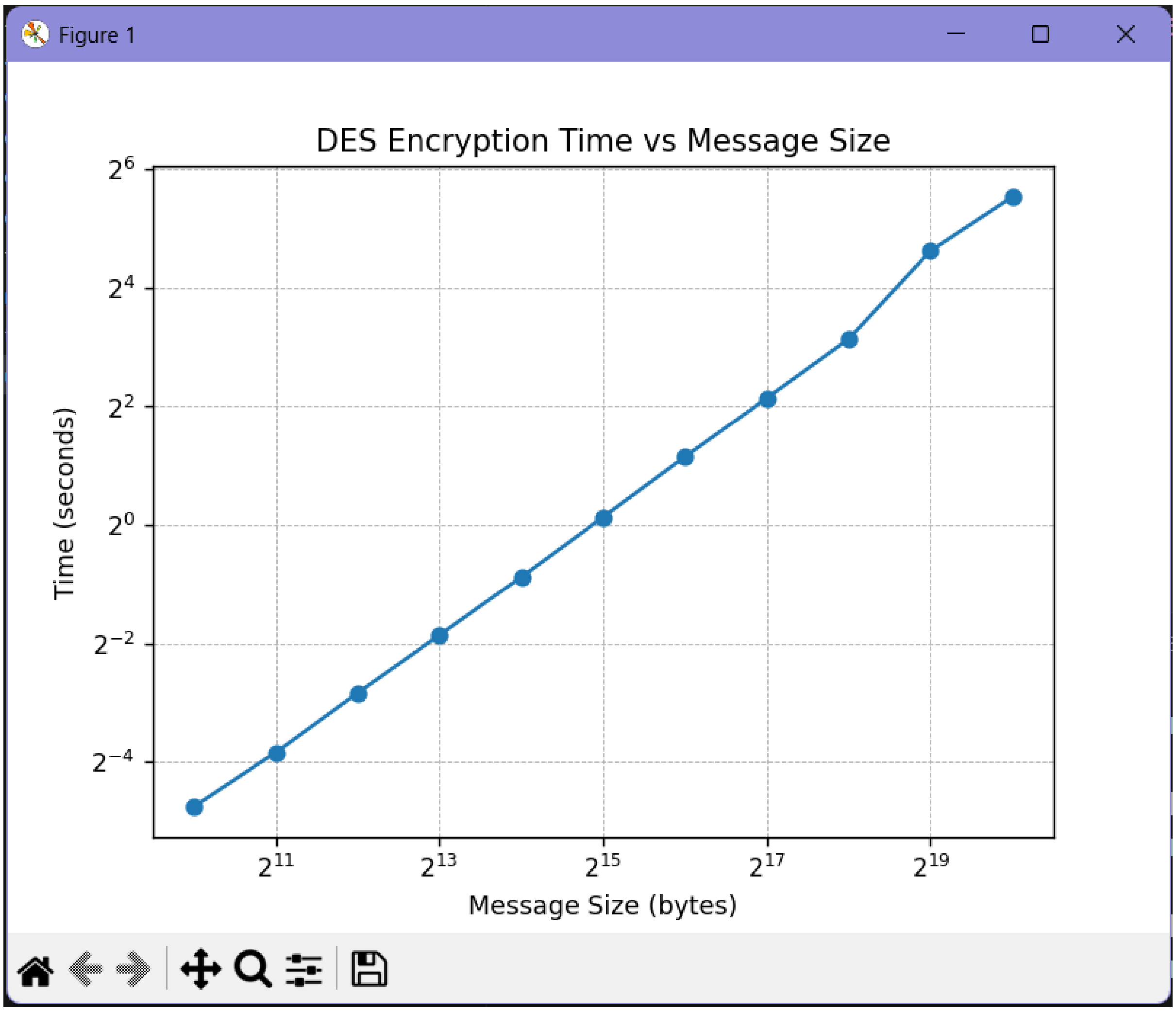
1. We created the feistel_implementation file that contains our implementation of the feistel protocol. It features:
   - an f function that applies a XOR bitwise operation
   - Encoding to or decoding from hex values from an arbitrarily large input string
   - Adaptive halving of the input depending on its byte length.

2. We created the derive_keys function that accepts a master_key (a seed) and the number of wanted keys and returns a list of k keys derived from the master_key in a deterministic way. The purpose of this function is to provide us with a seamless way to generate an arbitrary number of round keys for the sake of testing. We are aware that perfectly deterministic keys generated from deriving them from a constant master_key is not secure, but for the purpose of testing our algorithm, this method is good enough.

   The algorithm encrypts and decrypts arbitrarily large strings.

3. The DES implementation is contained in the des.py file. It follows all instructions and works for arbitrarily large strings.

4. We performed a performance analysis to find the time complexity of our newly built implementation of the DES algorithm. We quickly noticed that it takes a long time to reach 2^{26} bytes, so we cut the benchmark short, believing our current sample size was representative enough of the time complexity. Here's the graph illustrating just that:



We can see that the DES function runtime increases proportionally to the message size, meaning that we can guess that its time complexity is in the order of $O(n)$. This time complexity is normal for any cryptographic function, because it must iterate over all bytes at least once. In that regard, DES is like other cryptographic functions.
The constant factor observed in this benchmark what around the order of ~0.3 milliseconds per kilobyte, or ~300 milliseconds per megabyte of message size.