1. Affine Cipher

There are 312 possible distinct key combinations, with 12 numbers less than 26 coprime with 26 (1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, and 25) and 26 distinct possibilities for b (1...26). 12*26=312 possible distinct keys.

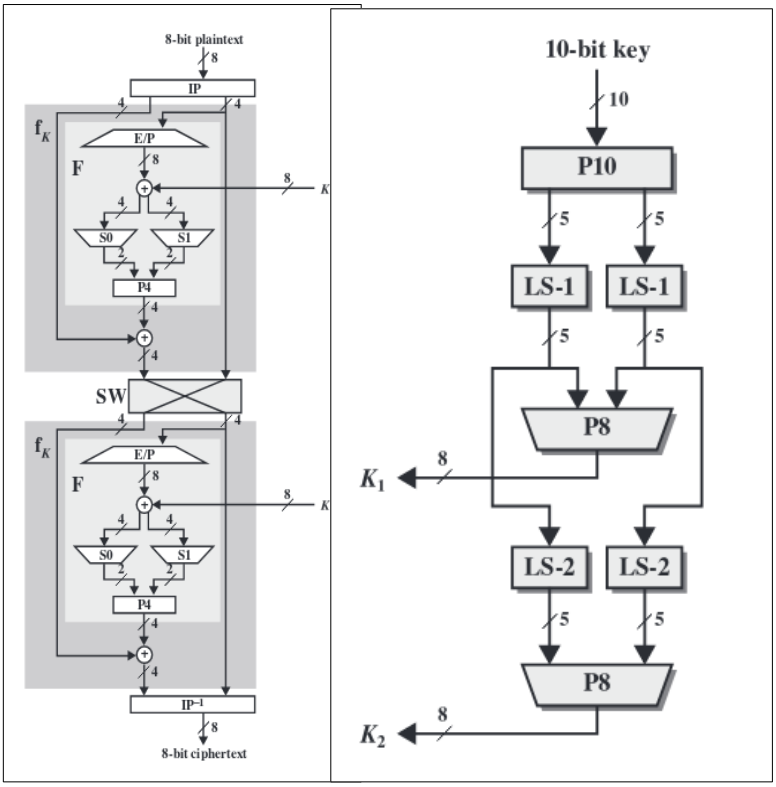
See attached affine test (testAffine.sh) for example. It may be configured with any test file and set of eligible keys to produce the correct encrypted and decrypted output.

It can be proven that the result of decrypting an encrypted text will result in the original text by substituting the equations of encryption and decryption into a nested pair, then simplifying the result, as shown below. This demonstrates that decrypting an encrypted message *x* with the same pair of keys *a*, *b* will result in the original message, modulo the number of symbols *m*; so long as *a* and *b* are coprime.

$$\begin{aligned}\mathrm{D}(\mathrm{E}(x)) &= a^{-1}(\mathrm{E}(x) - b) \bmod m \\ &= a^{-1}(((ax + b) \bmod m) - b) \bmod m \\ &= a^{-1}(ax + b - b) \bmod m \\ &= a^{-1}ax \bmod m \\ &= x \bmod m.\end{aligned}$$

My implementation of the affine cipher checks for each character if it is between ‘a’ and ‘z’ or ‘A’ and ‘Z’ and if not, simply applies no encryption/decryption to it before copying it to the output.

2. S-DES Encryption



Depicted to the left is the process of encrypting a message with S-DES, the structure of which my program mirrors quite closely in its implementation.

Decryption follows the same process, though with the usage of *k1* and *k2* in **F** switched.

With a 10bit key of all 1s (1023) as the input, *k1* and *k2* are generated both with values of all 1s (255). As now *k1 = k2*, this fails to be a form of two-way encryption– as the processes of encryption and decryption are now identical. This is known as a weak key,and would be true for any 10bit key which generates a key pair where *k1 = k2*.

3. Additional Questions

- Describe possible threats in information transmission and which type threat you can overcome with DES.

  Possible threats concerning information transmission generally relate to the confidentiality of the information being transmitted, or the reliability of the transmission method. DES can be used to help improve confidentiality by protecting somewhat against man-in-the-middle attacks so long as the key is communicated safely first and the message is encrypted before transmitted and only decrypted on the other end.

- Explain what you have done for source coding in Question 2.

  Text data is loaded from the file as a sequence of chars – which are interpreted in C as signed integers. My implementation casts these to unsigned integers before beginning the encryption process. As my implementation manipulates the bits of the unsigned integer directly at all stages of the process no further source coding is required. This also has improved performance over implementations that store individual bits differently as CPU “AND” and “OR” instructions can be used natively.

- What will you do for coding in information transmission? Use one illustration of example to explain it.

  Before a message is transmitted, it is wise to first encrypt or otherwise disguise it to outside parties. DES or some other encryption format may be used for this purpose.

- Is it true that “the higher the information rate is, the better the coding performance will be”? Justify your answer.

  The speed at which information may be transferred relies on multiple factors, including the medium of transmission and the systems involved in the transmission process. One factor in this process is the rate at which information can be coded into ciphertext before transmission, and so the faster the performance of the coding process, the less it restricts the maximum information rate. If there is a larger bottleneck elsewhere in the process then improvements in coding performance will have no effect on the overall bandwidth of information transfer.

References

Affine proof working used from https://en.wikipedia.org/wiki/Affine_cipher. Accessed 18-4-2019.

Simplified-DES process diagrams and specification for implementation sourced from http://mercury.webster.edu/aleshunas/COSC%205130/G-SDES.pdf. Accessed 18-4-2019.

Permute functions implementation based on operations generated using http://programming.sirrida.de/calcperm.php. Accessed 18-4-2019.