CS387 - Applied Cryptography

Ángel Sola Orbaiceta

December 2021

1 Concepts

Given a message $m \in M$, where M is the set of all possible messages, and a key $k \in K$, where K is the set of all possible keys, an encryption function E can be defined as:

$$E : M \times K \to C$$

where $c \in C$ is the *ciperthext* (being C the set of all possible ciphertexts). Conversely, a decryption function D can be defined as:

$$D : C \times K \to M$$

The **correctness property** states that, for all messages and keys, decrypting the result of encrypting a message must result in the message itself. Mathematically:

$$\forall m, k : D_k(E_k(m)) = m$$

The **security property** states that the ciphertext reveals nothing about the key or original message.

1.1 One-Time Pad

The one-time pad is based in the XOR ($\oplus$) function. The XOR function satisfies the property that any value XOR-ed with itself equals zero: $x \oplus x = 0$. The one-time pad uses this property so that, by using a key that's the same size as the ciphertext, we can do:

$$c = m \oplus k$$
$$m = c \oplus k$$

The one-time pad encryption and decryption functions are implemented in the *source/one_time_pad.py* file.

1.2 Perfect Cipher

Recall that given two events A and B in the same probability space, the **conditional probability** of B given that A occured is:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Now, given two messages, m and m^* drawn from the set of messages M are encrypted using a key $k \in K$ to produce a cypertext $c \in C$, a **perfect cipher** must hold that:

$$P\left(m = m^* | E_k\left(m\right) = c\right) = P\left(m = m^*\right)$$

We can now prove that the one-time pad is a perfect cipher as follows. For the one-time pad:

$$P\left(E_k\left(m\right) = c\right) = \sum_{m_i \in M} \sum_{k_i \in K} \frac{P(E_{k_i}(m_i) = c)}{|M| \times |K|} = \frac{|M|}{|M| \times |K|} = \frac{1}{|K|}$$

and, assuming a uniform distribution for the key space:

$$P\left(m = m^* \cap E_k(m) = c\right) = P(m = m^*) \times P(k = k^*) = \frac{P(m = m^*)}{|K|}$$

Therefore:

$$P\left(m = m^* | E_k(m) = c\right) = \frac{\frac{P(m=m^*)}{|K|}}{\frac{1}{|K|}} = P(m = m^*)$$

which means that the ciphertext reveals nothing about the key or original message.

Shanon's theorem If a cipher is perfect, it must be impractical ($|K| \geq |M|$).

2 Application of Symmetric Ciphers

Kolmogorov complexity The complexity K of a sequence s ($K(s)$) is the length of the shortest possible description of s. s is random if $K(s) = |s| + C$. The Kolmogorov complexity is uncomputable.

A Pseudo-Random Number Generator (PRNG) produces a long sequence of seemingly random bytes given an initial seed value. In Linux, *dev/random* can be used as a randomness pool.

2.1 Modes Of Operation

We assume the message we want to encrypt or decrypt can be broken into n blocks of size b:

$$m = m_0, m_1, m_2, \ldots, m_{n-1}$$

Electronic Codebook Mode each block is encrypted independently from each other:

$$c_i = E_k(m_i)$$

and decryption:

$$m_i = D_k(c_i)$$

The problem with ECB mode is that it doesn't hide repetition: equal blocks encrypt to equal ciperthexts. To run a cipher using the ECB mode of operation:

```
$ py symmetric/block_cli.py -e -m ecb -f <file> -k <key>
```

Cipher Block Chaining Mode the output of each block is XOR-ed with the input to the next block. Encryption:

$$c_i = E_k(m_i \oplus c_{i-1})$$

and if an **initialization vector** is used, the first block:

$$c_0 = E_k(m_0 \oplus IV)$$

and decryption:

$$m_i = D_k(c_i) \oplus c_{i-1}$$
$$m_0 = D_k(c_0) \oplus IV$$

To run a cipher using the CBC mode of operation:

```
$ py symmetric/block_cli.py -e -m cbc -f <file> -k <key>
```