12 Final thoughts

12.1 Block cipher modes

We have discussed block ciphers, and how to encrypt blocks of a specified size (64 bit for DES, or 128 bit for AES). **Block cipher modes** specify how to encrypt larger plaintexts.

Let E_k be the encryption routine of a block cipher with block size n bit. As a first step, we split a plaintext m into blocks $m = m_1 m_2 m_3 \ldots$ such that each m_i is n bits (we may have to pad).

Example 206. (ECB, shouldn't be used) In the simplest mode, known as **electronic codebook**, we just encrypt each plaintext block individually:

$$c_j = E_k(m_j)$$

The ciphertext is $c = c_1 c_2 c_3 \ldots$. Decryption simply computes $D_k(c_j) = m_j$.

Though natural, ECB has several severe weaknesses. Can you think of some?

Solution. Using ECB is nothing else but a classical substitution cipher, except that ECB operates on larger blocks. Just like a classical substitution cipher is vulnerable to frequency attacks, ECB leaves patterns in the ciphertext. For a striking visual example when encrypting a picture, see:

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

If a block repeats later in the message (or in a later message), it will be encrypted the same way. Hence, Eve can notice such repetitions. This is problematic in practice, for instance, because certain files always begin with the same blocks, so that Eve has a good chance of detecting the file type.

Also, knowing the filetype, Eve might be able to rearrange the ciphertext blocks to adjust the message. She can also attempt to delete certain ciphertext blocks.

Conclusion. Unless you know exactly why (e.g. sending already randomized messages), you should not use ECB.

Example 207. (CBC) In **cipherblock chaining** mode, we encrypt each plaintext block after chaining it with the previous cipherblock; that is:

$$c_j = E_k(m_j \oplus c_{j-1})$$

In order to do that for $j = 1$, we need a value for c_0, known as an **initialization vector** IV.

(a) How does decryption work?

(b) Why does the value IV need to be random?

Solution.

(a) The ciphertext is $c = c_0 c_1 c_2 c_3 \ldots$
Since $c_j = E_k(m_j \oplus c_{j-1})$, we have $D_k(c_j) = m_j \oplus c_{j-1}$ or $m_j = D_k(c_j) \oplus c_{j-1}$.
For instance. $m_1 = D_k(c_1) \oplus c_0$

(b) The value IV needs to be random, so that messages starting with the same plaintext block have different ciphertext blocks.

Comment. A way of transmitting $c_0 = \mathrm{IV}$ is by prepending a random plaintext block m_0 to m. The decryptor then just chooses a random IV and discards the first plaintext block.

There exist many other modes, including modes which already include features like authentication. Other common basic modes such as OFB or CTR turn the block cipher into a stream cipher (one advantage of that is that we don't need to encrypt full blocks at a time).

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Comment. One issue of ECB and CBC is the need for padding. If not handled properly, this can be exploited by a **padding oracle attack**:

https://en.wikipedia.org/wiki/Padding_oracle_attack

Example 208. Consider a block cipher with 4 bit block size and 4 bit key size such that

$$E_k(b_1b_2b_3b_4) = (b_2b_3b_4b_1) \oplus k.$$

(a) Encrypt $m = (0000\ 1011\ 0000\ \ldots)_2$ using $k = (1111)_2$ and ECB mode.

(b) Encrypt $m = (0000\ 1011\ 0000\ \ldots)_2$ using $k = (1111)_2$ and CBC mode ($\mathrm{IV} = (0011)_2$).

Solution. $m = m_1m_2m_3\ldots$ with $m_1 = 0000$, $m_2 = 1011$ and $m_3 = 0000$.

(a) $c_1 = E_k(m_1) = 0000 \oplus 1111 = 1111$
$c_2 = E_k(m_2) = 0111 \oplus 1111 = 1000$
Since $m_3 = m_1$, we have $c_3 = c_1$. Hence, the ciphertext is $c = c_1c_2c_3\ldots = (1111\ 1000\ 1111\ \ldots)$.

(b) $c_0 = 0011$
$c_1 = E_k(m_1 \oplus c_0) = E_k(0000 \oplus 0011) = E_k(0011) = 0110 \oplus 1111 = 1001$
$c_2 = E_k(m_2 \oplus c_1) = E_k(1011 \oplus 1001) = E_k(0010) = 0100 \oplus 1111 = 1011$
$c_3 = E_k(m_3 \oplus c_2) = E_k(0000 \oplus 1011) = E_k(1011) = 0111 \oplus 1111 = 1000$
Hence, the ciphertext is $c = c_0c_1c_2c_3\ldots = (0011\ 1001\ 1011\ 1000\ \ldots)$.

Comment. Clearly, our cipher is not meant to be secure. One damning issue (besides the short key and block size) is that it is linear (in both the plaintext and the key). Recall that AES introduces nonlinearity by replacing something like the 4 bit $b_2b_3b_4b_1$ with its inverse in $\mathrm{GF}(2^4)$.

12.2 Primes

For applications like RSA, we need the ability to produce large random primes.

Theorem 209. (prime number theorem) Up to x, there are roughly $x/\ln(x)$ many primes.

Let $\pi(x)$ be the exact number of primes up to x. The precise statement of the prime number theorem is that

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln(x)} = 1.$$

Some data. The proportion of primes up to 10^6 is $\frac{78,498}{10^6} = 7.85\%$. The estimate is $\frac{1}{\ln(10^6)} = \frac{1}{6\ln(10)} = 7.24\%$.

As we increase x, the proportion of primes must go down (why?!). For instance, the proportion of primes up to 10^{10} is $\frac{455,052,511}{10^{10}} = 4.55\%$. The estimate is $\frac{1}{\ln(10^{10})} = 4.34\%$.

The prime number theorem now makes it easy to extend that estimate as follows:

The proportion of primes up to 10^{100} is roughly $\frac{1}{\ln(10^{100})} = 0.434\%$.

The proportion of primes up to 10^{1000} is roughly $\frac{1}{\ln(10^{1000})} = 0.0434\%$. And so on.

Important conclusion. This means that, among 1000-digit numbers (base 10), about 1 of 2000 are prime! These numbers have about 3300 bits, which is more than currently needed for RSA (2048 bit recommended). Consequently, we can generate the random primes p, q simply by generating random numbers with about 1024 bit each and continuing until we have found two primes.