Block ciphers (and DES in particular)

We now introduce block ciphers at the example of **DES** (short for data encryption standard).

This sketch only provides an overview but does not include all details. See Chapter 4 in our book for these internals and detailed diagrams.

DES was the first public cryptosystem. While a public standard, the design decisions have been kept secret.

1974: proposed by IBM (lead by Horst Feistel; Lucifer) with input from NSA (key size reduced from 128 to 56 bits)
1976–2000: US national standard (broken by exhaustive search in 1997)
2000: replaced with AES (Rijndael) by NIST; however, **3DES** still considered secure (more later)

Why was the design secret? For many years, a particular mystery about DES was the choice of the S-boxes. Much later, in 1990, Biham and Shamir discovered **differential cryptanalysis**, a general method for breaking block ciphers. Surprisingly, it turned out that the particular choice of S-boxes made DES rather resistant against that attack. Indeed, as confirmed later, the IBM researchers had already discovered and anticipated that attack, but were asked by the NSA to keep it secret (it was a powerful weapon against other cryptosystems).

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

Comment. As our discussion will show, DES was designed to be implemented in hardware.

General principles of block cipher design

A block cipher takes a plaintext block of, say, B bits and encrypts it into a ciphertext block of B bits.

For instance, for DES, $B = 64$: 64 bit blocks are encrypted to 64 bit blocks.

For now, we will just focus on encrypting a single block.

However, we will need to talk about how to use a block cipher to encrypt longer plaintexts that need to be broken into many blocks (it is generally a bad idea to individually and independently encrypt each block).

The design of a block cipher is almost an art, but there are two guiding principles due to Claude Shannon, the father of information theory:

- confusion

refers to making the relationship between the ciphertext and the key as complex and involved as possible

(for instance, changing one bit of the key should change the ciphertext completely)

For instance. In DES, confusion is increased by the S-box substitutions. These are the only nonlinear part of DES. Without them, DES would be easily broken with linear algebra.

- diffusion

refers to dissipating the statistical structure of plaintext over the bulk of ciphertext

(for instance, changing one bit of the plaintext should change the ciphertext completely; likewise, changing one bit of the ciphertext should change the plaintext completely)

For instance. In DES, diffusion is increased by the E-box and P-box permutations.

Example 109. The classical substitution cipher provides only confusion.

Diffusion is completely missing. Changing bits of the plaintext only changes corresponding parts of the ciphertext. That's why frequency analysis can break these ciphers so easily.

Typical block ciphers are built by iteration, and consist of several **rounds**. Each round should have steps to increase both confusion and diffusion.

- **(key expansion)** First, we need to expand key k into several **round keys** $k_1, k_2, ..., k_n$ (n rounds).
  For instance. For DES, each round key k_i has 48 bits, which are drawn from the 56 bit DES key k in such a way that each bit of k shows up in about 14 of the 16 rounds.
- **(round functions)** Then, the message m is encrypted successively with R_{k_1}, R_{k_2}, ..., R_{k_n} to obtain c in the end.

$$m \to \boxed{R_{k_1}} \to \boxed{R_{k_2}} \to \ldots \to \boxed{R_{k_n}} \to c$$

  Each R_k is called a **round function**.
  For instance. For DES, there are $n = 16$ rounds; for AES-128, there are $n = 10$ rounds

A specific block cipher now needs specific algorithms for key expansion and round functions.

For DES, 16 rounds are used, which are identical in functionality but use different round keys k_i.

There is one additional, cryptographically irrelevant, step for DES: namely, there is a (fixed) **initial permutation** IP, which shuffles the bits of m before being sent to R_{k_1}. Similarly, the output of $R_{k_{16}}$ is shuffled with IP^{-1}, the inverse permutation, to produce c. (For the exact permutation see Chapter 4.4 in our book.)
Why? When implemented in hardware, this permutation does not cost any work, since it is just a wiring of the bits. In fact, the permutation somehow simplified the electrical engineering in the chips of the 70s.

A block cipher design: Feistel ciphers

Many ciphers, including DES (but not AES) are Feistel ciphers. This means that the encryption functions R_{k_i} are of a special format. The crucial ingredient is a **round function** $f_{k_i}(x)$.

This round function can be **any** function, such that x and $f_{k_i}(x)$ have the same size in bits (though only good choices will provide security). Also, several different round functions can be used for the different rounds.

To encrypt m using R_{k_i} (for DES, m is 64 bits and the round key k_i is 48 bits):

- Split the plaintext m into two halves (L_0, R_0) (for DES, each half is 32 bits).
- $L_1 = R_0$
  $R_1 = L_0 \oplus f_{k_i}(R_0)$
- Then, $R_{k_i}(m)$ is (L_1, R_1).

Example 110. How to decrypt one round? That is, how to obtain (L_0, R_0) from (L_1, R_1)?

Solution. First, $R_0 = L_1$. Then, $L_0 = R_1 \oplus f_{k_i}(R_0)$.
Important comment. In particular, we can take any round function f in the sense that we obtain some cipher, which can actually be decrypted (however, most choices for f will be insecure; see example below).
Comment. In hardware, the circuit for decryption is the same as for encryption, just reversed.

Example 111. What happens if we choose $f_{k_i}(R) = 0$ as the round function?

Solution. In that case, we are just swapping left and right half. No security whatsoever.

To finish the description of DES, we need to specify $f_{k_i}(R)$, where R is 32 bits and k_i is 48 bits.

We did that in class, but do not reproduce the description and diagrams here. See Chapter 4.4 of our book.
The crucial ingredients are an E-box (expansion), eight S-boxes (substitution) and a P-box (permutation).

Further comments on DES

The S-boxes $S_1, S_2, \ldots, S_8$ are lookup tables (for each 6 bit input, they specify a 4 bit output).

- They have been carefully designed.
  For instance, their design already anticipated and protected against differential cryptanalysis (which wasn't publicly known at the time).
- On the other hand, they do not follow any simple rule. In particular, they must not be linear (or close to it). If they were, DES would be entirely insecure.
  [Slightly more specifically, if the S-boxes were linear, then the encryption map $m \mapsto c$ would be linear. In the usual spirit of linear algebra, a few (m, c) pairs would then suffice to recover the key.]
- They are also designed so that if one bit is changed in the input, then at least 2 bits of the output change.
  Important consequence. Go through one application of the round function $f_{k_i}(R)$, and convince yourself that flipping one bit of R has the effect of flipping at least two bits of $f_{k_i}(R)$. Repeating this for 16 rounds, you can see how the goal of diffusion seems to be achieved: changing one bit of the plaintext should change the ciphertext completely.

Example 112. Sometimes it is stated that DES works with a 64 bit key size. In that case, every 8th bit is a parity bit, but the algorithm really operates with 56 bit keys.

Comment. Apparently, the NSA was interested in strengthening DES against any attack (recall that developments like differential cryptanalysis were foreseen) except brute-force. Indeed, the NSA seems to have pushed for a key size of 48 bits versus proposed 64 bits, and the result was a compromise for 56 bits.

Example 113. If DES is insecure because of its 56 bit key size, why not just increase that?

Solution. DES was designed specifically for that key size. Increasing it necessitates a completely new analysis on how to choose the S-boxes and so on.

On the other hand. See the upcoming discussion of 3DES for how to leverage the original DES to increase the key size.

However. With the advent of powerful successors like AES there are very few reasons to use 3DES for new cryptosystems. (One slight advantage of 3DES is its particular small footprint in hardware implementations.)

Example 114. Can we (easily) break DES if we know one of the round keys?

Solution. Absolutely! Recall that each round key consists of 48 bits taken from the overall 56 bit DES key. Hence, we know all but 8 bits of the key. We just need to brute-force these $2^8 = 256$ many possibilities.

Example 115. To (naively) brute-force DES, how much data must we encrypt?

Solution. By brute-forcing, we mean that, given a pair of 64-bit blocks m, c, we go through all 2^{56} possibilities (DES uses 56-bit keys) for k and look for k such that $E_k(m) = c$? We need to encrypt 2^{56} times 64 bits.

This is $2^{56} \cdot 8 = 2^{59}$ byte, or 512 pebibyte (binary analog of petabyte) or 576 petabyte (since $2^{59} \approx 5.76 \cdot 10^{17}$).

How long will this take? Of course, this depends on your machine. Assume we are able to encrypt 1 GB/sec. Then, this will take us about $5.76 \cdot 10^8$ sec, or about 18.3 years.

Of course, such a brute-force attack can be fully parallelized to quickly bring this number down to less than an hour for a powerful attacker. Also, the attack can be sped up considerably by careful design (like early aborts).

For comparison. Though mostly of theoretical value, for DES, some possibilities for attacks better than brute-force are known: for instance, as of 2008, linear cryptanalysis can mount an attack with 2^{43} known plaintexts in about 2^{40} (instead of 2^{56}) steps.

Example 116. (bonus challenge) Using DES, are there blocks m, c such that $E_k(m) = c$ for more than one key k?

I don't know the answer and couldn't find it easily. Maybe you are more skilled?

Example 117. (3DES) A simple approach to increasing the key size of DES, without the need to design and analyze a new block cipher, is **3DES**. It consists of three applications of DES to each block and is still considered secure.

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

The 3DES standard allows three keying options:

- k_1, k_2, k_3 independent keys: $3 \times 56 = 168$ key size, but effective key size is 112
- $k_1 = k_3$: $2 \times 56 = 112$ key size, effective key size is stated as 80 by NIST
- $k_1 = k_2 = k_3$: this is just the usual DES, and provides backwards compatibility (which is a major reason for making the middle step a decryption instead of another encryption).

Comment. The reason for the reduced effective key sizes is the meet-in-the-middle attack. It is also the reason why something like 2DES is not used. See next example!

Comment. NIST approved 3DES until 2030 for sensitive government data.

Example 118. (no 2DES) Explain why "2DES" does not really provide extra security over DES.

Solution. Let's denote DES encryption with E_k and decryption with D_k. The keys k are 56 bits.

Then, 2DES encrypts according to $c = E_{k_2}(E_{k_1}(m))$. The key size of 2DES is $56 + 56 = 112$ bits.

- A brute-force attack would go through all possibilities for pairs (k_1, k_2), of which there are $2^{56} \cdot 2^{56} = 2^{112}$, to check whether $c = E_{k_2}(E_{k_1}(m))$. That requires 2^{112} 2DES computations.
- On the other hand, note that $c = E_{k_2}(E_{k_1}(m))$ is equivalent to $D_{k_2}(c) = E_{k_1}(m)$.

  Assuming sufficient memory, we first go through all 2^{56} keys k_2 and store the values $D_{k_2}(c)$ in a lookup table.

  We then go through all 2^{56} keys k_1, compute $E_{k_1}(m)$ and see if we have stored that value before. (Even though this is a huge table, the cost for checking whether an element is in the table can be disregarded; thanks to the magic of hash tables!)

  Comment. In this second step, we see that m and c should be more than one block (otherwise we get too many candidate keys $k = (k_1, k_2)$).

  The total number of DES computations to break 2DES therefore is $2^{56} + 2^{56} = 2^{57}$, which is hardly more than for breaking DES!

This is known as a meet-in-the-middle attack.

https://en.wikipedia.org/wiki/Meet-in-the-middle_attack

Comment. The price to pay is that this attack also requires memory for storing This sort of approach is referred to as a time-memory trade-off. Instead of brute-forcing 2DES in 2^{112} steps, we can attack it in 2^{57} steps while storing 2^{56} values of the size of m.

Comment. This applies to any block cipher, not just DES!

Comment. For some block ciphers it is the case that for all pairs of keys k_1, k_2, there is a third key k_3 such that $E_{k_2}(E_{k_1}(m)) = E_{k_3}(m)$. In that case, we say that the cipher is a group, and double (or triple, or quadruple) encryption does not add any additional security! DES, however, is not a group.

Example 119. Explain why 3DES, used with three different keys, only has effective key size 112.

Solution. (fill in the details!) Instead of going through all k_1, k_2, k_3 to check whether

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

(which would take $2^{56} \cdot 2^{56} \cdot 2^{56} = 2^{168}$ DES computations), we can use that the latter is equivalent to

$$D_{k_3}(c) = D_{k_2}(E_{k_1}(m)).$$

Now proceed as in the previous example ... to see that we can break 3DES with 2^{112} DES computations. How much memory do we need?

Example 120. (extra; use as PRG) ANSI X9.17 is a U.S. federal standard for a PRG based on 3DES.

Input: random, secret 64 bit seed s, key k for 3DES (keying option 2)

Produce a random number as follows:

- obtain current time D, compute $t = 3\mathrm{DES}_k(D)$
- output $x = 3\mathrm{DES}_k(s \oplus t)$ (that's the pseudo-random output)
- update the seed to $s = 3\mathrm{DES}_k(x \oplus t)$ for future use

Comment. ANSI (American National Standards Institute) X9 are standards for the financial industry.

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

Comment. The same approach can be applied to any block cipher.

Comment. It is common practice to add in time for PRGs that are used to generate enormous amounts of data. If nothing else, it slightly increases the entropy and reduces the likelihood of "short" periods.

Example 121. (extra; DES-X) To increase the key size of DES, the following variation, known as DES-X, was proposed by Ron Rivest in 1984:

$$c = k_3 \oplus \mathrm{DES}_{k_2}(m \oplus k_1)$$

What is the key size of DES-X? What about the effective key size?

Solution. k_1 and k_3 are 64 bit, while k_2 is 56 bits. That's a total key size of 184 bits for DES-X.

However, just like for 3DES, proceeding as in a meet-in-the-middle-attack (without the need of much storage) reduces the effective key size to at most $184 - 64 = 120$ bits.

Comment. This approach of xoring with a subkey before and after everything else is known as **key whitening**. This features in many modern ciphers, including AES.

https://en.wikipedia.org/wiki/DES-X

Block cipher modes

Block ciphers 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 122. (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 123. (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.

The ciphertext is $c = c_0 c_1 c_2 c_3 \ldots$ (that's one more block than for the plaintext $m = m_1 m_2 m_3 \ldots$).

(a) How does decryption work?

(b) Why should the value IV be unpredictable (e.g. be chosen randomly)?

Solution.

(a) 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 should be unique, so that messages starting with the same plaintext block have different ciphertext blocks. More generally, it should be unpredictable so that Eve cannot mount a chosen-plaintext attack to test if an earlier plaintext equals her guess. See Example 125.

Just checking. What would happen if we set $c_j = E_k(m_j) \oplus c_{j-1}$ instead? In that case, we would gain nothing over ECB: since Eve knows all c_j, she can compute $c_j \oplus c_{j-1} = E_k(m_j)$.

Comment. CBC makes random access possible during decryption (but not encryption). That means, we don't need to decrypt $c = c_0 c_1 c_2 c_3 \ldots$ sequentially but can directly decrypt $c_N c_{N+1} \ldots$ for some random N.

Example 124. Consider the (silly) 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).

Extra. In each case, can you decrypt c to get back the original m?

Example 125. (BEAST attack) BEAST is short for Browser Exploit Against SSL/TLS and was brought to public attention in 2011. The attack is based on the fact that the IV used by SSL was obtained from a previous ciphertext block (instead of randomly!):

Scenario. Imagine that plaintext blocks $m_1m_2\ldots$ are continuously being encrypted using CBC to cipherblocks. However, the plaintexts are from different parties and Eve can ask for her own plaintexts to be encrypted along the way.

In such a scenario, different plaintexts should be separately encrypted using CBC, meaning that a new random IV should be chosen each time.

Eve's goal. Suppose Eve has observed the ciphertext blocks c_{j-1}, c_j and her goal is to find out whether $m_j = x$ where x is her educated guess. Obviously, this is something that Eve should not be able to do!

The exploit. Because the IV for the next encryption is c_j, and because Eve can interject plaintext blocks to be encrypted for her, she can ask for $m_{j+1} = x \oplus c_{j-1} \oplus c_j$ (these are all known to Eve!) to be encrypted next.

Because CBC with IV c_j is used, this results in $c_{j+1} = E_k(m_{j+1} \oplus c_j) = E_k(x \oplus c_{j-1})$.

Eve can now compare this with $E_k(m_j \oplus c_{j-1}) = c_j$ (which she knows!) to find out whether $m_j = x$.

https://en.wikipedia.org/wiki/Transport_Layer_Security#BEAST_attack

There exist many other modes, including modes which already include features like authentication. Other common basic modes such as OFB (output feedback) or CTR (counter) 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