CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 5
January 25, 2010

Data encryption standard (DES)

The Data Encryption Standard is a block cipher that operates on 64-bit blocks and uses a 56-bit key.

It became an official Federal Information Processing Standard (FIPS) in 1976. It was officially withdrawn as a standard in 2005 after it became widely acknowledged that the key length was too short and it was subject to brute force attack.

Nevertheless, triple DES (with a 112-bit key) is approved through the year 2030 for sensitive government information.

The new Advanced Encryption Standard (AES), based on the Rijndael algorithm, became an official standard in 2001. AES supports key sizes of 128, 192, and 256 bits and works on 128-bit blocks.

Feistel networks

DES is based on a *Feistel network*.

This is a general method for building an invertible function from any function f that scrambles bits.

It consists of some number of stages.

- Each stage i maps a pair of n-bit words (L_i, R_i) to a new pair (L_{i+1}, R_{i+1}). ($n = 32$ in case of DES.)
- By applying the stages in sequence, a t-stage network maps (L_0, R_0) to (L_t, R_t).
- (L_0, R_0) is the plaintext, and (L_t, R_t) is the corresponding ciphertext.

DES Feistel network

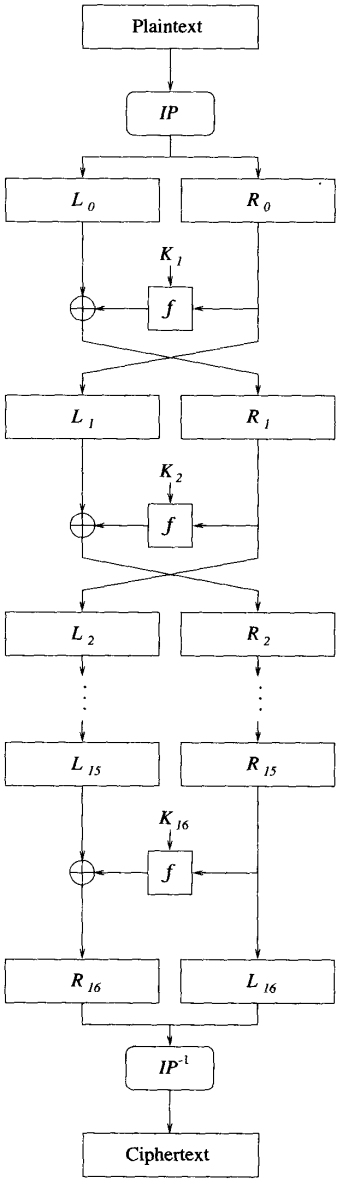

Figure 4.4: The DES Algorithm.

One stage

Each stage works as follows:

$$L_{i+1} = R_i \tag{1}$$

$$R_{i+1} = L_i \oplus f(R_i, K_i) \tag{2}$$

Here, K_i is a *subkey*, which is generally derived in some systematic way from the master key k.

The inversion problem is to find (L_i, R_i) given (L_{i+1}, R_{i+1}). Equation 1 gives us R_i. Knowing R_i and K_i, we can compute $f(R_i, K_i)$. We can then solve equation 2 to get

$$L_i = R_{i+1} \oplus f(R_i, K_i)$$

Properties of Feistel networks

The *security* of a Feistel-based code lies in the construction of the function f and in the method for producing the subkeys K_i.
The *invertibility* follows just from properties of $\oplus$ (exclusive-or).

DES uses a 16 stage Feistel network.
The pair L_0R_0 is constructed from a 64-bit message by a fixed initial permutation IP.
The ciphertext output is obtained by applying IP^{-1} to $R_{16}L_{16}$.

The scrambling function $f(R_i, K_i)$ operates on a 32-bit data block and a 48-bit key block. Thus, $48 \times 16 = 768$ key bits are used.

They are all derived in a systematic way from the 56-bit primary key and are far from independent of each other.

Obtaining the subkey

The scrambling function $f(R_i, K_i)$ is the heart of DES.

It operates on a 32-bit data block and a 48-bit key block (called a *subkey*.

The 56-bit master key k is split into two 28-bit pieces C and D. At each stage, C and D are rotated by one or two bit positions. Subkey K_i is then obtained by applying a fixed permutation (transposition) to CD.

The scrambling function

At the heart of the scrambling function are eight "S-boxes" that compute Boolean functions with 6 binary inputs $c_0, x_1, x_2, x_3, x_4, c_1$ and 4 binary outputs y_1, y_2, y_3, y_4.
Thus, each computes some fixed function in $\{0,1\}^6 \to \{0,1\}^4$.

Special property of S-boxes: For fixed values of (c_0, c_1), the resulting function on inputs $x_1, \ldots, x_4$ is a permutation from $\{0,1\}^4 \to \{0,1\}^4$.

Can regard an S-box as performing a substitution on four-bit "characters", where the substitution performed depends both on the structure of the particular S-box and on the values of its "control inputs" c_0 and c_1.

The eight S-boxes are all different and are specified by tables.

DES scrambling network

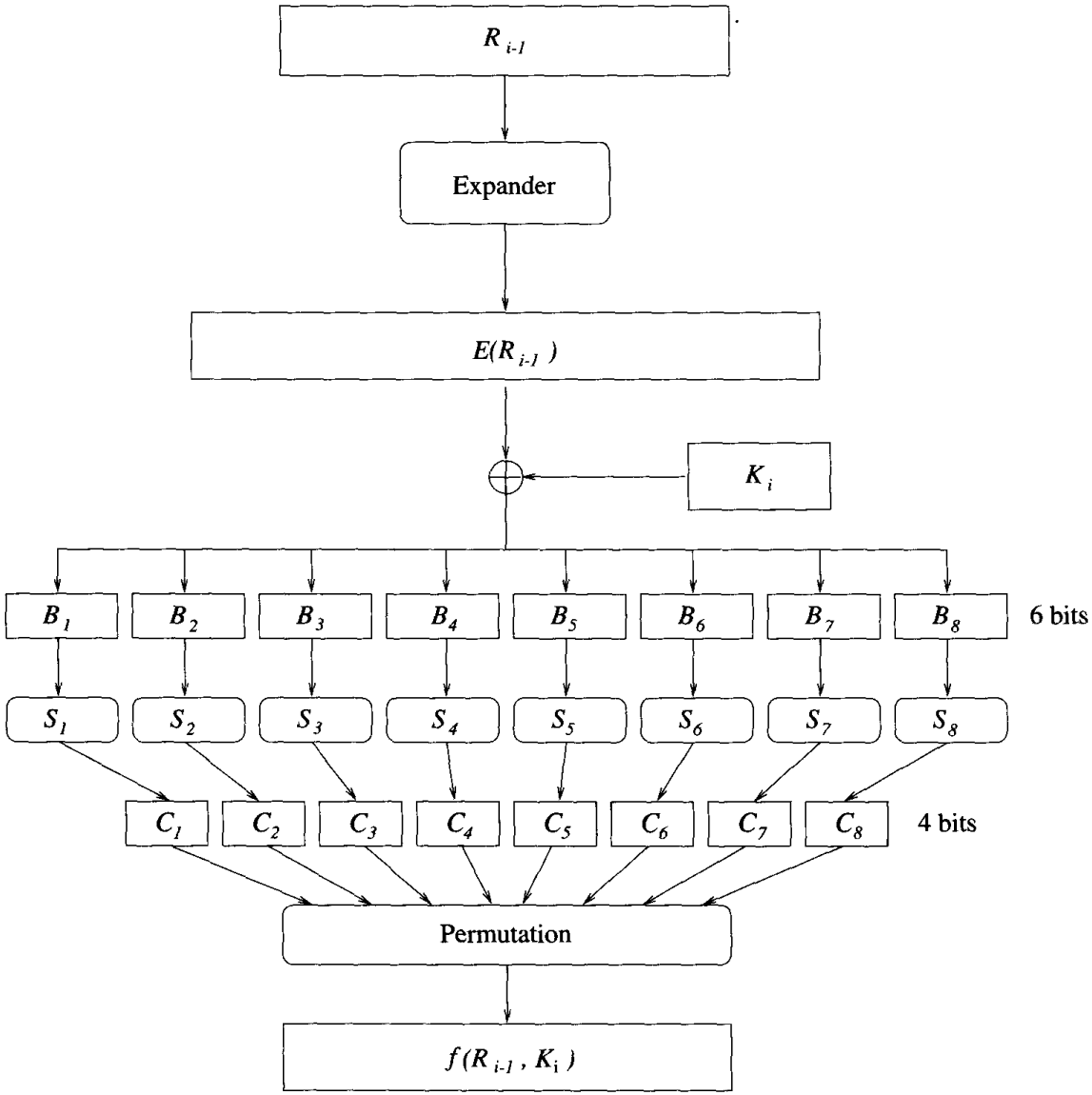

Figure 4.5: The DES Function $f(R_{i-1}, K_i)$.

Connecting the boxes

The S-boxes together have a total of 48 input lines.

Each of these lines is the output of a corresponding $\oplus$-gate.

- One input of each of these $\oplus$-gates is connected to a corresponding bit of the 48-bit subkey K_i. (This is the only place that the key enters into DES.)
- The other input of each $\oplus$-gate is connected to one of the 32 bits of the first argument of f.

Since there are 48 $\oplus$-gates and only 32 bits in the first argument to f, some of those bits get used more than once.

The mapping of input bits to $\oplus$-gates is called the *expansion permutation* E.

Expansion permutation

The expansion permutation connects input bits to $\oplus$ gates. We identify the $\oplus$ gates by the S-box inputs to which they connect.

- Input bits $32, 1, 2, 3, 4, 5$ connect to the six $\oplus$ gates that go input wires $c_0, x_1, x_2, x_3, x_4, c_1$ on S-box 1.
- Bits $4, 5, 6, 7, 8, 9$ are connect to the six $\oplus$ gates that go input wires $c_0, x_1, x_2, x_3, x_4, c_1$ on S-box 2.
- The same pattern continues for the remaining S-boxes.

Thus, input bits $1, 4, 5, 8, 9, \ldots 28, 29, 32$ are each used twice, and the remaining input bits are each used once.

Connecting the outputs

The 32 bits of output from the S-boxes are passed through a fixed permutation P (transposition) that spreads out the output bits.

The outputs of a single S-box at one stage of DES become inputs to several different S-boxes at the next stage.

This helps provide the desirable "avalanche" effect, where a change in one input bit spreads out through the network and causes many output bits to change.

Security considerations

DES is vulnerable to a brute force attack because of its small key size.

However, it has turned out to be remarkably resistant to recently-discovered (in the open world) sophisticated attacks.

Differential cryptanalysis: Can break DES using "only" 2^{47} chosen ciphertext pairs.

Linear cryptanalysis: Can break DES using 2^{43} chosen plaintext pairs.

Neither attack is feasible in practice.

Double DES

Double DES is simply double encryption where the base cryptosystem is DES.

Even if the original system is not a group (and DES is not), double encryption does not result in a cryptosystem with twice the effective key length because of another "birthday"-style known-plaintext attack.

A birthday attack on double DES

Start with a known plaintext-ciphertext pair (m, c).

- Encrypt m and decrypt c using *all possible DES keys*.
- We are guaranteed of finding at least one match, since if (k_1, k_2) is the real key pair, then $E_{k_1}(m) = D_{k_2}(c)$.
- A matching pair is *good* if it defines the same encryption function as the real key pair. Otherwise it is *bad*.
- We expect roughly $\sim 2^{48}$ bad pairs.
- Try each matching pair on a second plaintext-ciphertext pair.
- Now we expect roughly 2^{-16} bad key pairs, so with high probability, any of the remaining pairs are good.
- Effective key length is then just 2^{57}, not hoped-for 2^{112}.

(See Trapp & Washington, §4.7, and Katz & Lindell, pp. 182–184.)

A plausibility argument why double DES does not help

Why do we believe the number of bad pairs is so small?

- For each key k, E_k maps m and D_k maps c to elements of $\mathcal{M} = \mathcal{C}$. (Recall $|\mathcal{M}| = |\mathcal{C}| = 2^{64}$.)
- For a key pair (k_1', k_2') the probability that $E_{k_1'}(m) = D_{k_2'}(c)$ would be 2^{-64}, *assuming* $E_{k_1'}(m)$ and $D_{k_2'}(c)$ are uniformly and independently distributed over $\mathcal{M}$.
- There are 2^{112} key pairs, so the expected number of matching pairs would be $2^{112} \cdot 2^{-64} = 2^{48}$.
- Each matching key pair has probability 2^{-64} of working with a second plaintext-ciphertext pair (again assuming randomness).
- The expected number of key pairs that work for both plaintext-ciphertext pairs is $2^{48} \cdot 2^{-64} = 2^{-16}$.

DES as a function of the key

We know the randomness assumptions are not quite correct:

- The pair (m, c) was itself not chosen at random.
- We know little about how E_k and D_k depend on k.

We do know the expected number of key pairs surviving both plaintext-ciphertext pairs is at least 1, since the real key pair is such a pair.

I don't know how to make this argument completely rigorous, but it is plausible that it will work to break DES.

This alone is reason enough to seriously doubt the security of double DES.

Reference: Ralph C. Merkle and Martin E. Hellman, "On the security of multiple encryption", *Commun. ACM 24*, 7 (1981), 465–467.

Extreme example

Here's an extreme example to show why one can't simply assume the birthday attack will succeed.

Assume a cryptosystem like DES except that the encryption function is as follows:

$$E_k(m) = \begin{cases} m & \text{if } k \text{ is even} \\ \sim m & \text{if } k \text{ is odd (where } \sim \text{ means bitwise complement)} \end{cases}$$

Now, when one tries the birthday attack, *exactly half* of all key pairs (k_1, k_2) match.

Repetitions with a new plaintext-ciphertext pair encoded by the same real key pair do not reduce the number of matches further.

Triple DES

Three key triple DES (3TDES) uses three DES encryptions, i.e., $E_{k_3}(E_{k_2}(E_{k_1}(m)))$, giving it an actual key length of 168 bits.

3TDES can be broken (in principle) by a known plaintext attack using about 2^{90} single DES encryptions and 2^{113} steps.

While this attack is still not practical, the effective security thus lies in the range between 90 and 113, which is much smaller than the apparent 168 bits.[1]

[1] The *effective security* is the base-2 logarithm of the amount of work to break a cryptosystem. This is the same as the amount of work to carry out a brute force attack on a cryptosystem with keys of that length. Thus, a cryptosystem that can be broken in time 2^{90} is said to have 90-bit effective security.

Variants of triple DES

Several variants have been proposed.

TDES-EDE uses a decryption in the middle step instead of an encryption, i.e., $E_{k_3}(D_{k_2}(E_{k_1}(m)))$.
Security is the same as for 3TDES.
One advantage: TDES-EDE encryption and decryption functions can be used for single DES by taking $k_1 = k_2 = k_3$ equal to the single DES key.

2TDES uses only two keys k_1 and k_2. It is the same as 3TDES where $k_3 = k_1$. Known plaintext attacks or chosen plaintext attacks reduce the effective security to only 80 bits.[2]

[2] See Wikipedia for further information on triple DES.

Block ciphers

Recall: *Block ciphers* map b-bit plaintext blocks to b-bit ciphertext blocks.

Block ciphers typically operate on fairly long blocks, e.g., 64-bits for DES, 128-bits for Rijndael (AES).

Block ciphers can be designed to resist known-plaintext attacks, provided b is large enough, so they can be pretty secure, even if the same key is used to encrypt a succession of blocks, as is often the case.

[What goes wrong if b is small?]

Using a block cipher

One rarely wants to send messages of exactly the block length.

To use a block cipher to encrypt arbitrary-length messages:

- Divide the message into blocks of size b.
- Pad the last partial block according to a suitable padding rule and/or add another block at the end.
- Use the block cipher in some chaining mode to encrypt the sequence of resulting blocks.

Padding

Padding extends the message to satisfy two requirements:

- The length must be a multiple of b.
- It must be possible to recover the exact original message from the padded message.

Just sticking 0's on the end of a message until its length is a multiple of b will not satisfy the second requirement.

A padding rule must describe about how much padding was added.

Padding rules

Here's one rule that works.

- Let $r = |m| \bmod b$. This is the size of the last (partial) block if $|m|$ is not a multiple of b.
- Pad each message with p 0's followed by a length ℓ binary representation of p.
- Choose $p = (-r - \ell) \bmod b$. The padded message length is $\lfloor |m|/b \rfloor \cdot b + r + p + \ell \equiv 0 \pmod{b}$.

To unpad, interpret the last ℓ bits of the message as a binary number p; then discard a total of $p + \ell$ bits from the right end of the message.

Padding example

Example, $b = 64$:

- At most 63 0's ever need to be added, so a 6-bit length field is sufficient.
- A message m is then padded to become $m' = m \cdot 0^p \cdot \overline{p}$, where $\overline{p}$ is the 6-bit binary representation of p.
- p is chosen so that $|m'| = |m| + p + 6$ is a multiple of 64.

Chaining mode

A *chaining mode* tells how to encrypt a sequence of plaintext blocks $m_1, m_2, \ldots, m_t$ to produce a corresponding sequence of ciphertext blocks $c_1, c_2, \ldots, c_t$, and conversely, how to recover the m_i's given the c_i's.

Electronic Codebook Mode (ECB)

Each block is encrypted separately.

- To encrypt, Alice computes $c_i = E_k(m_i)$ for each i.
- To decrypt, Bob computes $m_i = D_k(c_i)$ for each i.

This is in effect a monoalphabetic cipher, where the "alphabet" is the set of all possible blocks and the permutation is E_k.

Cipher Block Chaining Mode (CBC)

Prevents identical plaintext blocks from having identical ciphertexts.

- To encrypt, Alice computes the XOR of the current plaintext block with the previous ciphertext block.
  That is, $c_i = E_k(m_i \oplus c_{i-1})$.
- To decrypt, Bob computes $m_i = D_k(c_i) \oplus c_{i-1}$.

To get started, we take $c_0 = \mathrm{IV}$, where IV is a fixed *initialization vector* which we assume is publicly known.