CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 6
September 18, 2017

Symmetric Cryptosystem Components

Padding

- Bit padding
- Byte padding

Data Encryption Standard (DES)

Symmetric Cryptosystem Components

Building blocks

Symmetric (one-key) ciphers combine simple ideas, some of which we've already seen:

- Substitution
- Transposition
- Composition
- Subkey generation
- Chaining

Substitution: Replacing one letter by another

The methods discussed so far are based on letter substitution.

The Caesar cipher *shifts* the alphabet cyclically.
This yields 26 possible permutations of the alphabet.

In general, one can use any permutation of the alphabet, as long as we have a way of computing the permutation and its inverse.
This gives us 26! possible permutations.

Often, permutations are specified by a table called an *S-box*.

Transposition: Rearranging letters

Another technique is to rearrange the letters of the plaintext.

`this message is encoded with a transposition cipher`

1. Pick a number: 9.
2. Write the message in a 9-column matrix (ignoring spaces):

```
thi sme ssa
gei sen cod
edw ith atr
ans pos iti
onc iph er
```

3. Read it out by columns[1]

`tgeao hednn iiwsc ssipi metop enhsh scaie sottr adri`

[1]Spaces are not part of ciphertext.

Composition: Building new ciphers from old

Let (E', D') and (E'', D'') be ciphers.
Their *composition* is the cipher (E, D) with keys of the form $k = (k'', k')$, where

$$E_{(k'',k')}(m) = E''_{k''}(E'_{k'}(m))$$

$$D_{(k'',k')}(c) = D'_{k'}(D''_{k''}(c)).$$

Can express this using *functional composition*.
$h = f \circ g$ is the function such that $h(x) = f(g(x))$.

Using this notation, we can write $E_{(k'',k')} = E''_{k''} \circ E'_{k'}$ and $D_{(k'',k')} = D'_{k'} \circ D''_{k''}$.

Subkey generation

When ciphers are composed, each component cipher needs a key called a *subkey*. Together, those subkeys can get rather large and unwieldy.

For practical reasons, the subkeys are themselves often generated by a deterministic process dependent on a *master key*, which is the user key of the resulting cryptosystem.

Questions: How are subkeys generated in:

- the Caesar cipher?
- the Vigenère cipher?
- the Enigma machines?
- the one-time pad?

Chaining modes

A *chaining mode* describes how to employ the cipher on a sequence of blocks.

One obvious way is to repeatedly use the cipher with the same key on each successive block. This is called *Electronic Code Book (ECB)* mode.

We can improve on this by generating a different subkey for each block.

For example, successive subkeys might depend on the block number, as in simple stream ciphers, or also on previous plaintext and/or ciphertext.

Other ways of using the cipher

Other chaining modes use the block cipher in other ways than simply to encrypt blocks.

Output Feedback Mode (OFM) encrypts using XOR (like the one-time pad) and uses the block cipher instead to generate a sequence of subkeys for the XOR.

- The first subkey is the encryption of a fixed *initialization vector* (IV).
- Each successive subkey is the encryption of the previous one.

We'll look at some widely used chaining modes later.

Padding

Padding

Block ciphers are designed to handle sequences of blocks.

To send a message m of arbitrary length, it must first be *encoded* into a new message m' that consists of a sequence of blocks.

m' is then encrypted, transmitted, and decrypted.

After decrypting, m' must be *decoded* to recover the original message m.

A *padding rule* describes the encoding and decoding process.

How to pad

An obvious padding rule is to append 0's to the end of m until its length is a multiple of the block length b.

Unfortunately, this can't be properly decoded, since the receiver does not know how many 0's to discard from m'.

Condition: A padding rule must describe how much padding was added.

Suggestions?

Some easy padding rules

1. Pad with 0's, then *prepend* a block containing the length of m.

   Example: $b = 8, m = 01011, m' = \overbrace{00000101}^{\text{5 in binary}}\ 01011000$.
   Drawback: Must know the length of m before beginning.

2. Pad with 0's, then *append* a block containing the length of m.

   Example: $b = 8, m = 01011, m' = 01011000\ \overbrace{00000101}^{\text{5 in binary}}$.
   Drawback: Wasteful of space

3. Pad with a single 1 bit followed by 0's.
   Example: $b = 8, m = 01011, m' = 01011100$.
   Drawback: Need to count bits.

What happens if the length of m is already a multiple of b?

Compact bit padding

Here's a padding rule that is both space efficient and easy to decode.

- Choose $\ell = \lceil \log_2 b \rceil$. This is the number of bits needed to represent (in binary) any number in the interval $[0 \ldots (b-1)]$.
- Choose p as small as possible so that $|m| + p + \ell$ is a multiple of b.
- Pad m with p 0's followed by a length ℓ binary representation of p.

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.

Bit padding

For arbitrary bit strings, append p 0's to the message followed by an ℓ-bit number whose value is p.

ℓ must be big enough to represent any value for p between 0 and $b-1$, so choose $\ell = \lceil \log_2 b \rceil$.

Choose p so that $|m| + p + \ell$ is a multiple of b.

The padded message is $m \cdot 0^p \cdot \overline{p}$, where $\overline{p}$ is the binary representation of p.

Bit patting examples

1. $b = 8, m = 01011, m' = 01011\overbrace{000}^{0}.$
2. $b = 8, m = 1110, m' = 11100\,\overbrace{001}^{1}.$
3. $b = 8, m = 111, m' = 11100\,\overbrace{010}^{2}.$
4. $b = 8, m = 010110, m' = 01011000\ \ 00000\,\overbrace{111}^{7}.$

Bit padding on 64-bit blocks

- 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 as small as possible so that $|m'| = |m| + p + 6$ is a multiple of 64.

$$\overbrace{\boxed{\quad m \quad}}^{|m|}\overbrace{\boxed{0\ldots0}}^{p}\overbrace{\boxed{\overline{p}}}^{6}$$

Block codes on byte strings

Often messages and blocks consist of a sequence of 8-bit bytes.

In that case, padding can be done by adding an integral number of bytes to the message.

At least one byte is always added to avoid ambiguity.

PKCS7 padding

PKCS7 #7 is a message syntax described in internet RFC 2315.

- Fill a partially filled last block having k "holes" with k bytes, each having the value k when regarded as a binary number.
- If $k = 0$, an empty block is added before padding.

Example: Block length = 8 bytes.
m = "`hello`".
$m' =$ `68 65 6C 6C 6F` 03 03 03.

On decoding, if the last block of the message does not have this form, then a decoding error is indicated.

Example: The last block cannot validly end in . . . 25 00 03.

What is the last block if $k = 0$?

Possible information leakage from padding

Suppose Alice uses AES (block length 128) in ECB mode to send 129-bit messages.

Eve has a plaintext-ciphertext pair (m', c') and intercepts a new cipher text c for an unknown message m.

Because of padding, both c and c' are two blocks long. Let c_2 and c_2' be the second blocks of each, respectively.

Then the last bit of m is the same as the last bit of m' iff $c_2 = c_2'$, so Eve learns the last bit of m.

Data Encryption Standard (DES)

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 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.

A Feistel network

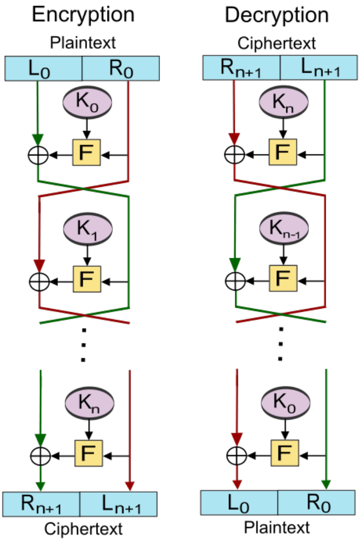

Image credit: Wikipedia: Feistel cipher, http://en.wikipedia.org/wiki/Feistel_cipher

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, and f is the scrambling function (shown as F in the diagram).

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 scrambling function f and in the method for producing the subkeys K_i.

The *invertibility* follows just from properties of $\oplus$ (exclusive-or).

DES Feistel network

DES uses a 16 stage Feistel network.

The pair $L_0 R_0$ 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.

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

S-boxes

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

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.

Each computes some fixed function in $\{0,1\}^6 \to \{0,1\}^4$.

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

Special properties 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 $\{0,1\}^4 \rightarrow \{0,1\}^4$.

Hence, 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.

Thus, DES's 8 S-boxes are capable of performing 32 different substitutions on 4-bit fields.

DES scrambling network

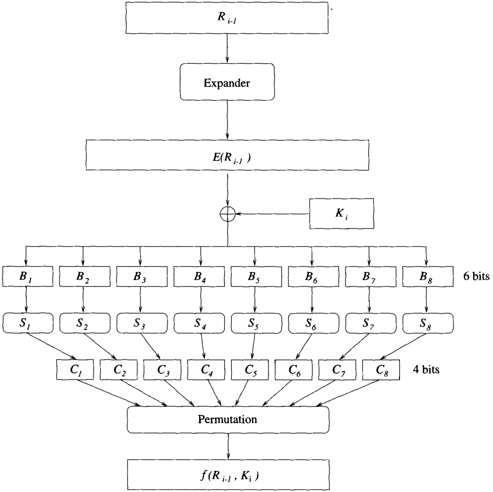

Figure 4.5: The DES Function $f(R_{i-1}, K_i)$.

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.

Obtaining the subkey

The scrambling function 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.

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.