CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 4
January 24, 2013

Block ciphers

Block ciphers

A *block cipher* is an encryption system where the base message space $\mathcal{M}_0$ is finite. Elements of $\mathcal{M}_0$ are called *blocks*.
Blocks are typically bit strings of some convenient length such as 64 or 128.

A block cipher can be used in *electronic codebook (ECB) mode* to encrypt arbitrarily long messages:

1. Represent message m as a sequence of blocks $b_1, b_2, \ldots, b_r$.
2. Encrypt each block using the base cipher, so $c_i = E_k(b_i)$, $i \in [1 \ldots r]$. The same key k is used for each.
3. Output the sequence $c_1, c_2, \ldots, c_r$ of encrypted blocks.

Analysis of the Caesar cipher

The Caesar cipher is an example of a *block cipher*, where the blocks are single letters.

Although the Caesar cipher is not practical because of its small key space, many of its properties are representative of any block cipher used in ECB mode.

We now explore its security properties.

Cryptanalysis

Breaking the Caesar cipher: An example

Suppose you intercept the ciphertext JXQ.
You quickly discover that $E_3(\text{GUN}) = \text{JXQ}$.
But is $k = 3$ and GUN is the correct decryption?

You then discover that $E_{23}(\text{MAT}) = \text{JXQ}$.
Now you are in a quandary. Which decryption is correct?

Have you broken the system or haven't you?

You haven't found the plaintext for sure, but you've reduced the possibilities down to a small set.

Breaking the Caesar cipher: Extending these ideas

The longer the correct message, the more likely that only one key results in a sensible decryption.

For example, suppose the ciphertext were "EXB JXQ".
We saw two possible keys for "JXQ" — 3 and 23.
Trying them both we get:

$$k = 3: \quad D_3(\text{EXB JXQ}) = \text{BUY GUN}.$$

$$k = 23: \quad D_{23}(\text{EXB JXQ}) = \text{HAE MAT}.$$

Latter is nonsense, so we know $k = 3$ and the message is "BUY GUN".

Breaking the Caesar cipher: Conclusion

Information-theoretic security of the Caesar cipher.

- $r = 1$: It is perfectly secure.
- $r > 1$: It is only partially secure or completely breakable, depending on message length and redundancy present in the message.

There is a whole theory of redundancy of natural language that allows one to calculate a number called the "unicity distance" for a given cryptosystem. If a message is longer than the unicity distance, there is a high probability that it is the only meaningful message with a given ciphertext and hence can be recovered uniquely, as we were able to recover "BUY GUN" from the ciphertext "EXB JXW" in the example. See [Sti06, section 2.6] for more information on this interesting topic.

Trying all keys

A *brute force attack* tries all possible keys k.

For each k, Eve computes $m_k = D_k(c)$ and tests if m_k is meaningful. If exactly one meaningful m_k is found, she knows that $m = m_k$.

Given long enough messages, the Caesar cipher is easily broken by brute force—one simply tries all 26 possible keys to see which leads to a sensible plaintext.

The Caesar cipher succumbs because the key space is so small.

Automating brute force attacks

With modern computers, it is quite feasible for an attacker to try millions ($\sim 2^{20}$) or billions ($\sim 2^{30}$) of keys.

The attacker also needs an automated test to determine when she has a likely candidate for the real key.

How does one write a program to distinguish valid English sentences from gibberish?

One could imagine applying all sorts of complicated natural language processing techniques to this task. However, much simpler techniques can be nearly as effective.

Random English-like messages

Consider random messages whose letter frequencies are similar to that of valid English sentences.

For each letter b, let p_b be the probability (relative frequency) of that letter in normal English text.

A message $m = m_1 m_2 \ldots m_r$ has probability $p_{m_1} \cdot p_{m_2} \cdots p_{m_r}$.

This is the probability of m being generated by the simple process that chooses r letters one at a time according to the probability distribution p.

Determining likely keys

Assume Eve knows that $c = E_k(m)$, where m was chosen randomly as described above and k is uniformly distributed.

Eve easily computes the 26 possible plaintext messages $D_0(c), ..., D_{25}(c)$, one of which is correct.

To choose which, she computes the conditional probability of each message given c, then picks the message with the greatest probability.

This guess will not always be correct, but for letter distributions that are not too close to uniform (including English text) and sufficiently long messages, it works correctly with very high probability.

How long should the keys be?

The DES (Data Encryption Standard) cryptosystem (which we will talk about shortly) has 56-bit keys for a key space of size 2^{56}.

A special DES Key Search Machine was built as a collaborative project by Cryptography Research, Advanced Wireless Technologies, and EFF. (Click here for details.)

This machine was capable of searching 90 billion keys/second and discovered the RSA DES Challenge key on July 15, 1998, after searching for 56 hours. The entire project cost was under $250,000.

Now, 13+ years later, the same task could likely be done on a commercial cluster computer such as Amazon's Elastic Compute Cloud (EC2) at modest cost.

What is safe today and into the future?

DES with its 56-bit keys offers little security today.

80-bit keys were considered acceptable in the past decade, but in 2005, NIST proposed that they be used only until 2010.

Triple DES (with 112-bit keys) and AES (with 128-bit keys) will probably always be safe from brute-force attacks (but not necessarily from other kinds of attacks).

Quantum computers, if they become a reality, would cut the effective key length in half (see Wikipedia "key size"), so some people recommend 256-bit keys (which AES supports).

Cryptography before computers

Large-scale brute force attacks were not feasible before computers.

While Caesar is easily broken by hand, clever systems have been devised that can be used by hand but are surprisingly secure.

Monoalphabetic ciphers

The Caesar cipher uses only the 26 rotations out of the 26! permutations on the alphabet. The *monoalphabetic cipher* uses them all. A key k is an arbitrary permutation of the alphabet. $E_k(m)$ replaces each letter a of m by $k(a)$ to yield c. To decrypt, $D_k(c)$ replaces each letter b of c by $k^{-1}(b)$.

The size of the key space is $|\mathcal{K}| = 26! > 2^{74}$, large enough to be moderately resistant to a brute force attack.

Nevertheless, monoalphabetic ciphers can be readily broken using letter frequency analysis, given a long enough message.

This is because *monoalphabetic ciphers preserve letter frequencies*.

How to break monoalphabetic ciphers

Each occurrence of a in m is replaced by $k(a)$ to get c.
Hence, if a is the most frequent letter in m, $k(a)$ will be the most frequent letter in c.

Eve now guesses that a is one of the most frequently-occurring letters in English, i.e., 'e' or 't'.
She then repeats on successively less frequent ciphertext letters.

Of course, not all of these guesses will be correct, but in this way the search space is vastly reduced.

Moreover, many wrong guesses can be quickly discarded even without constructing the entire trial key because they lead to unlikely letter combinations.

Building block ciphers

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*.

We can also expand the number of possible substitutions by grouping characters together into blocks and treating each block as a "letter" in an expanded alphabet.

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

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

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

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

Practical ciphers

Practical ciphers are built using many compositions of substitution and transposition ciphers.

Combined in this way, they can be quite effective.

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.

DES Feistel network

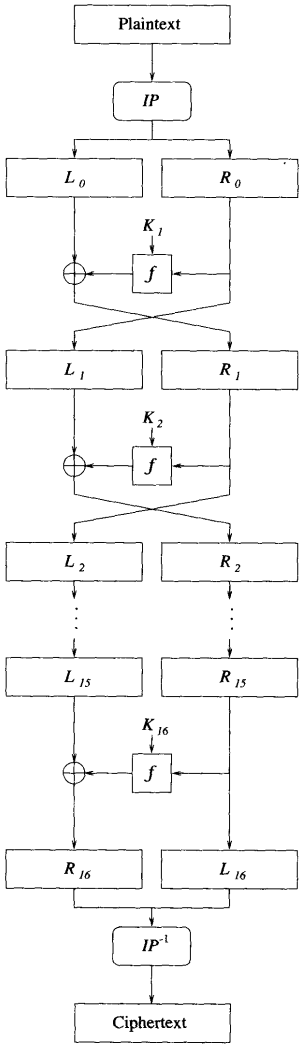


Figure 4.4: The DES Algorithm.

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 $\{0,1\}^6 \longrightarrow \{0,1\}^4$.

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 $\{0,1\}^4 \longrightarrow \{0,1\}^4$.

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

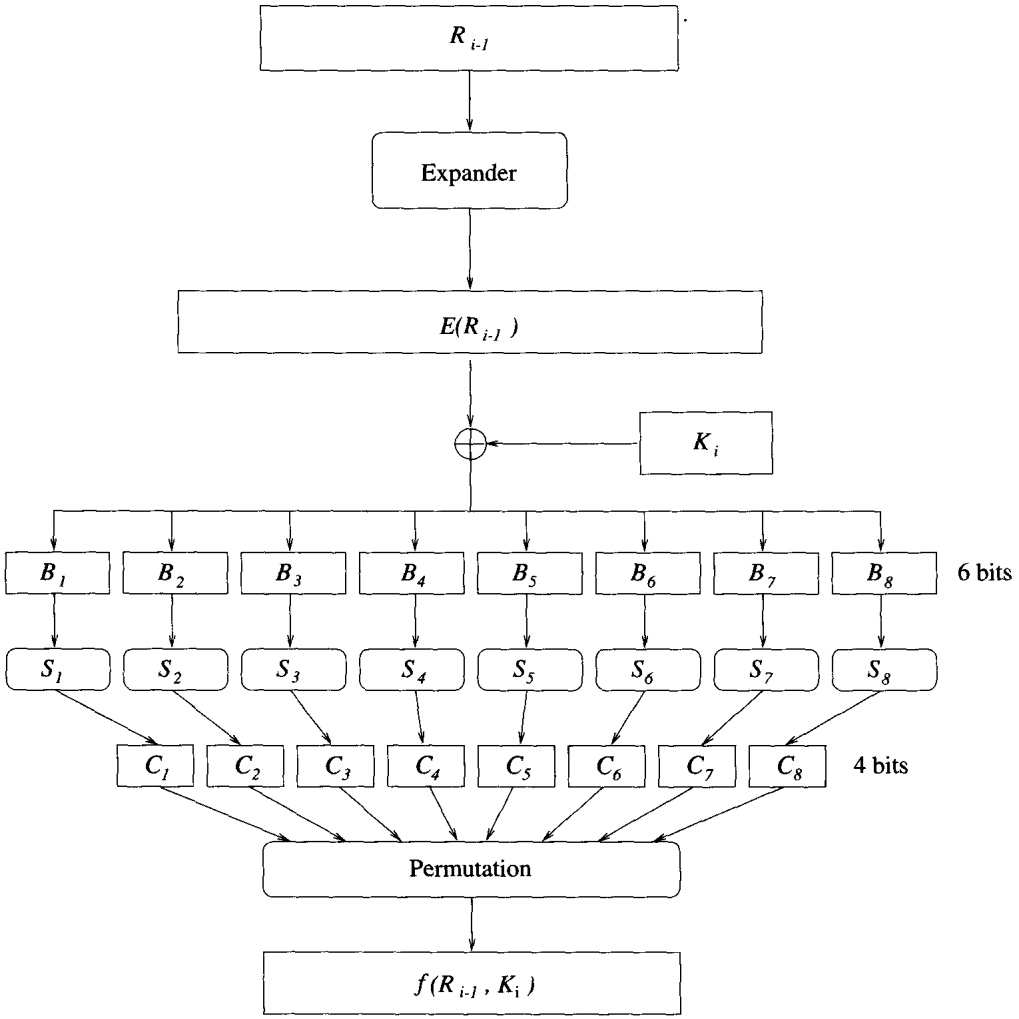


Figure 4.5: The DES Function $f(R_{i-1}, K_i)$.

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.

Using block ciphers

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 resulting sequence of 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.

Suggestions?

Padding rules

Here's one rule that works.

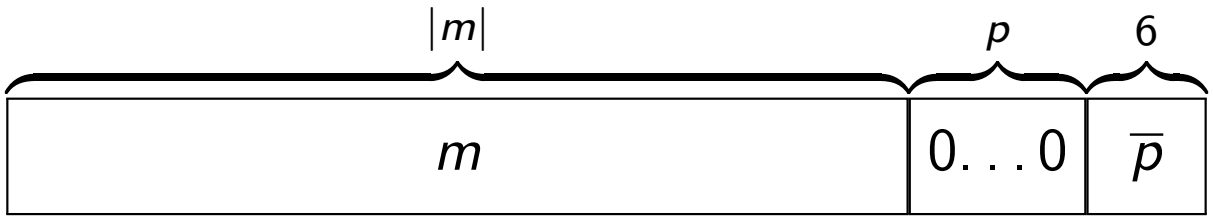
- Choose $\ell = \lceil \log_2 b \rceil$. This is the number of bits needed to represent (in binary) any number in the interval $[0 \ldots (b-1)]$.
- Choose p as small as possible so that $|m| + p + \ell$ is a multiple of b.
- Pad each message with p 0's followed by a length ℓ binary representation of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

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



References


Douglas R. Stinson.
Cryptography: Theory and Practice.
Chapman & Hall/CRC, third edition, 2006.
ISBN-10: 1-58488-508-4; ISBN-13: 978-58488-508-5.