Modern Block Ciphers

will now look at modern block ciphers
one of the most widely used types of cryptographic algorithms
provide secrecy and/or authentication services
in particular will introduce DES (Data Encryption Standard), the
most important one.
It will be replaced by AES (Advanced Encryption Standard)
Block vs Stream Ciphers

block ciphers process messages in blocks, each of which is then en/decrypted like a substitution on very big characters
- 64-bits or more
stream ciphers process messages a bit or byte at a time when en/decrypting
many current ciphers are block ciphers
hence are focus of course

Block Cipher Principles

most symmetric block ciphers are based on a Feistel Cipher Structure
needed since must be able to decrypt ciphertext to recover messages efficiently
block ciphers look like an extremely large substitution would need table of $2^{64}$ entries for a 64-bit block
Feistel proposed to approximate the large substitutions using idea of a product cipher
Shannon introduced the same idea in 1945
Claude Shannon and Substitution-Permutation Ciphers

in 1949 Claude Shannon introduced idea of substitution-permutation (S-P) networks
  • modern substitution-transposition product cipher
these form the basis of modern block ciphers
S-P networks are based on the two primitive cryptographic operations we have seen before:
  • substitution (S-box)
  • permutation (P-box)

provide confusion and diffusion of message

Confusion and Diffusion

cipher needs to completely obscure statistical properties of original message
a one-time pad does this
more practically Shannon suggested combining elements to obtain:

**diffusion** – the statistical structure of plaintext is dissipated into bulk of ciphertext. Each plaintext digit affects the value of many ciphertext digits. Eg. \( y_n = \sum_i^k m_{n+i} \mod 26 \) or using permutation followed by some function such as xor.

**confusion** – makes relationship between ciphertext and key as complex as possible. So make it hard to deduce the key. Eg. Substitution with a key.
Feistel Cipher Structure

Horst Feistel devised the **feistel cipher**
- based on concept of invertible product cipher
- partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves

implements Shannon’s substitution-permutation network concept

Input: 2w bits (L₀, R₀)
L₁ = R₀
R₁ = L₀ xor F(R₀, K₁)  K₁ is the first subkey
Expand to Lᵢ, Rᵢ
F does not need to be a reversible function!
Feistel Cipher Design Principles

Block size
- increasing size improves security, but slows cipher. 64 bit common. AES uses 128 bits

Key size
- increasing size improves security, makes exhaustive key searching harder, but may slow cipher. 128 bits now common.

Number of rounds
- increasing number improves security, but slows cipher. 16 rounds common.

Subkey generation
- greater complexity can make analysis harder, but slows cipher

Round function
- greater complexity can make analysis harder, but slows cipher

Fast software en/decryption & ease of analysis
- are more recent concerns for practical use and testing

Dr. Edwin Sha
DES History

IBM developed Lucifer cipher
- by team led by Feistel
- used 64-bit data blocks with 128-bit key
then redeveloped as a commercial cipher with input from NSA and others
in 1973 NBS issued request for proposals for a national cipher standard
IBM submitted their revised Lucifer which was eventually accepted as the DES

DES Design Controversy

although DES standard is public
was considerable controversy over design
- in choice of 56-bit key (vs Lucifer 128-bit)
- and because design criteria were classified
subsequent events and public analysis show in fact design was appropriate
DES has become widely used, esp in financial applications
Triple DES is more secured recommended by NIST
**DES Encryption**

Initial Permutation IP

First step to make the analysis complicated: diffusion. IP reorders the input data bits even bits permuted to Left half, odd bits to Right half quite regular in structure (easy in h/w)

Use 2 tables to define the permutation and inverse permutation. see text Table 3.2

example:

\[
\text{IP}(675a6967 \ 5e5a6b5a) = (ffb2194d \ 004df6fb)
\]
Single Round of DES

![Diagram of Single Round of DES Algorithm](image)

**DES Round Structure**

uses two 32-bit L & R halves

as for any Feistel cipher can describe as:

\[
L_i = R_{i-1}
\]

\[
R_i = L_{i-1} \text{ xor } F(R_{i-1}, K_i)
\]

F: takes 32-bit R half and 48-bit subkey and:
- expands R to 48-bits using expanded permutation E by adding 16 bits
- Adds (xor) to subkey
- passes through 8 S-boxes to get 32-bit result
- finally permutes this using 32-bit perm P

F does not need to be invertible. What happens if F always produces a constant 1.
**DES F function**

Substitution Boxes S

have eight S-boxes which map 6 to 4 bits

each S-box is actually a table with 4 rows and 16 columns
  - outer bits 1 & 6 select one rows
  - inner bits 2-5 select a column
  - Each selected entry will be the new 4 bits

From 8 S-boxes, result is 8 blocks of 4 bits, or 32 bits

Row selection depends on both data & key
  - feature known as autoclaving (autokeying)

example:

\[ S(18 \ 09 \ 12 \ 3d \ 11 \ 17 \ 38 \ 39) = 5fd25e03 \]
**DES Key Schedule**

forms subkeys used in each round consists of:

- initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
- 16 stages consisting of:
  - rotating each half separately either 1 or 2 bits depending on the key rotation schedule $K$
  - Give the shifted output to next round and F
  - permuting them by PC2 for use in function $f$, selecting 24-bits from each half

**DES Decryption**

decrypt must unwind steps of data computation with Feistel design, do encryption steps again using subkeys in reverse order (SK16 … SK1) note that IP undoes final IIP (inverse initial perm.) step of encryption

1st round with SK16 undoes 16th encrypt round ....

16th round with SK1 undoes 1st encrypt round then final IIP of decryption undoes initial encryption IP thus recovering original data value
Avalanche Effect

key desirable property of encryption algorithm
where a change of one input or key bit results in changing
approx half output bits
making attempts to “home-in” by guessing keys impossible
DES exhibits strong avalanche

Strength of DES – Key Size

56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
brute force search looked hard
recent advances have shown is possible
  • in 1997 on Internet in a few months
  • in 1998 on dedicated h/w (EFF) in a few days
  • in 1999 above combined in 22hrs!
still must be able to recognize plaintext
now considering alternatives to DES such as AES and triple DES
Strength of DES – Timing Attacks

attacks actual implementation of cipher
specifically use fact that calculations can take varying times
depending on the value of the inputs to it
It is not effective for symmetric ciphers because the computations
are not that hard and are quite uniform.
But might be useful for public key ciphers.
particularly problematic on smartcards

Strength of DES – Analytic Attacks

now have several analytic attacks on DES
these utilise some deep structure of the cipher
• by gathering information about encryptions
• can eventually recover some/all of the sub-key bits
• if necessary then exhaustively search for the rest
generally these are statistical attacks
include
• differential cryptanalysis
• linear cryptanalysis
Linear Cryptanalysis

recent development
also a statistical method
must be iterated over rounds, with decreasing probabilities
developed by Matsui et al in early 90's
based on finding linear approximations
can attack DES with $2^{47}$ known plaintexts, still in practice infeasible.
The known most effective approach

Simple Example
Linear Cryptanalysis

Lets use a simple SPN as an example. Four stages with 5 keys, \(k^1, k^2, k^3, k^4, k^5\), 4-bit S-box (a mapping from 4 bits to another 4 bits). Assume that all S are the same and known.

Stage 1: input \(x \rightarrow u^1 = x \text{xor} k^1 \rightarrow v^1 = \text{pass} \ S \text{ every 4 bits of} \ u^1 \rightarrow w^1 = \text{permutation of} \ v^1\)

Stage 2: input \(w^1 \rightarrow u^2 = w^1 \text{xor} k^2 \rightarrow v^2 = \text{pass} \ S \text{ every 4 bit of} \ u^2 \rightarrow w^2 = \text{permutation of} \ v^2\)

Stage 4: input \(w^3 \rightarrow u^4 = w^3 \text{xor} k^4 \rightarrow v^4 = \text{pass} \ S \text{ every 4 bit of} \ u^4 \rightarrow \text{Output} \ y = v^4 \text{xor} k^5\)

Idea: Try to find part of \(k^5\) by many known plaintext/ciphertext pairs
1. Given a guessed \(k^5\) and a known \(y\), it is easy to find the corresponding \(u^4\)
2. We try to find the prob of some input \(X\) xor some bits of \(u^4\). Then try MANY known plaintext/ciphertext pairs to see if a particular guessed key can give you this similar probability. Why: S-box is not totally random.

The probability and bias of xor

Assume \(X_1, X_2, X_3, \ldots\) are independent.

\(\Pr(X_i=0) = p_i, \quad \Pr(X_i=1) = (1-P_i)\)

\(\Pr(X_1 \text{xor} X_2 = 0) = p_1 p_2 + (1- p_1 ) (1- p_2 )\)

Define bias \(e_i = (p_i - \frac{1}{2}).\) \(e_i = 0\) means it is random

\(e_{1,2} = (1/2 + e_1) (1/2 + e_2) + (1/2 - e_1) (1/2 -e_2)-1/2 = 2 \ e_1 \ e_2\)

Pilling-up lemma: let \(e_{1,2,3,\ldots,k}\) denote the bias of the random variables \(X_1 \text{xor} X_2 \text{xor} \ldots \text{xor} X_K\). Then \(e_{1,2,3,\ldots,k} = 2^{k-1} \ (e_1 \times e_2 \times \ldots \times e_K)\).

It is interesting to see that as long as one X has bias \(=0\), then the bias of all will be 0. One way to construct a fair coin from \(K-1\) unfair coins.

It is important that we have all the \(X_i\) are bias in order to get a total bias \(\neq 0\). Yes, we can find them from I/O of S_box.
The S Box Example

\[
\begin{array}{cccc|cccc}
X1 & X2 & X3 & X4 & Y1 & Y2 & Y3 & Y4 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc|cccc}
X1 & X2 & X3 & X4 & Y1 & Y2 & Y3 & Y4 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[Pr(X1 \ xor \ X4 \ xor \ Y3 = 0) = \frac{3}{8}. \text{ We want to find some combination that is bias.} \]

But \[Pr(X2 \ xor \ Y2 \ xor \ Y4 = 0) = \frac{4}{16} = \frac{1}{4} \] Take advantage of these non-randomness. The bias is \( \frac{1}{4} - \frac{1}{2} = -\frac{1}{4}. \)

Linear Approx. Table

\[
\begin{array}{cccccccccccccccc}
\alpha & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F \\
\hline
0 & 16 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
1 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
2 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
3 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
4 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
5 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
6 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
9 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
A & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
B & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
C & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
D & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
E & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
F & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
\end{array}
\]

\((x1, x2, x3, x4) = a \ ; (y1, y2, y3, y4) = b; \text{ eg } a=(1011) \ b=(0100), \text{ we get 12.} \)

Bias (distance from \( \frac{1}{2} \)) = \((12-8)/16 = \frac{1}{4} \) It is a good choice because it is far from \( \frac{1}{2} \).
Pick some S-boxes

More calculations

This approximation incorporates four active S-boxes:
- In $S_3^1$, the random variable $T_1 = U_3^1 \oplus U_3^2 \oplus U_3^3 \oplus V_3^3$ has bias 1/4
- In $S_3^2$, the random variable $T_2 = U_3^2 \oplus V_3^2 \oplus V_3^3$ has bias $-1/4$
- In $S_3^3$, the random variable $T_3 = U_3^3 \oplus V_3^3 \oplus V_3^3$ has bias $-1/4$
- In $S_3^4$, the random variable $T_4 = U_{14}^3 \oplus V_{14}^3 \oplus V_{14}^3$ has bias $-1/4$

The four random variables $T_1, T_2, T_3, T_4$ have biases that are high in absolute value. Further, we will see that their exclusive-or will lead to cancellations of “intermediate” random variables.

How about $T_1 \text{xor} T_2 \text{xor} T_3 \text{xor} T_4$? We will assume they are independent (an approximation). This bias $= 2^3 (1/4) (-1/4)^3 = -1/32$. 
Substitute Variables

\[ T_1 = U_1^1 + U_7^1 + U_9^1 + V_3^1 = X_5 \oplus K_1^1 \oplus X_7 \oplus K_7^1 \oplus X_8 \oplus K_8^1 \oplus V_3^1 \]
\[ T_2 = U_2^1 + V_2^1 + V_3^1 = V_3^1 \oplus K_2^1 \oplus V_2^1 \oplus V_3^1 \]
\[ T_3 = U_0^1 + V_0^1 + V_3^1 = V_0^1 \oplus K_0^1 \oplus V_0^1 \oplus V_3^1 \]
\[ T_4 = U_1^4 + V_4^4 + V_{14}^4 = V_3^3 \oplus K_4^3 \oplus V_{14}^4 + V_{16}^4. \]

If we compute the x-or of the random variables on the right sides of the above equations, we see that the random variable
\[
X_5 \oplus X_7 \oplus X_8 \oplus V_0^1 \oplus V_0^1 \oplus V_3^1 \oplus V_{14}^3 \oplus V_{16}^3
\]
\[ \oplus K_1^1 \oplus K_7^1 \oplus K_8^1 \oplus K_2^1 \oplus K_3^1 \oplus K_1^4 \quad (3.1) \]
\[ V_0^3 = U_0^4 \oplus K_0^4 \]
\[ V_8^3 = U_{14}^4 \oplus K_{14}^4 \]
\[ V_{14}^3 = U_8^4 \oplus K_8^4 \]
\[ V_{16}^3 = U_{16}^4 \oplus K_{16}^4 \]

Now we substitute these four expressions into (3.1), to get the following:
\[
X_5 \oplus X_7 \oplus X_8 \oplus U_0^4 \oplus U_8^4 \oplus U_{14}^4 \oplus U_{16}^4
\]
\[ \oplus K_1^1 \oplus K_7^1 \oplus K_8^1 \oplus K_2^1 \oplus K_3^1 \oplus K_4^1 \oplus K_1^4 \oplus K_{14}^4 \oplus K_{16}^4 \quad (3.2) \]

The Probability must be

\[
K_1^1 \oplus K_7^1 \oplus K_8^1 \oplus K_0^2 \oplus K_6^3 \oplus K_{14}^4 \oplus K_0^4 \oplus K_8^4 \oplus K_{14}^4 \oplus K_{16}^4
\]

has the (fixed) value 0 or 1. It follows that the random variable
\[
X_5 \oplus X_7 \oplus X_8 \oplus U_0^4 \oplus U_8^4 \oplus U_{14}^4 \oplus U_{16}^4
\]
\[ (3.3) \]

has bias equal to \( \pm 1/32 \), where the sign of this bias depends on the values of unknown key bits. Note that the random variable (3.3) involves only plaintext bits and bits of \( u^4 \). The fact that (3.3) has bias bounded away from 0 allows us to carry out the linear attack mentioned at the beginning of Section 3.3.

Can find correct 8 bits (those 2 S-boxes) in \( k^5 \), by trying at least 8000 pairs of \( x \) and \( y \) (plaintexts/ciphertexts). Count which 8 bits will give the bias 1/32.
Differential Cryptanalysis

one of the most significant recent (public) advances in cryptanalysis
known by NSA in 70's in DES design but not published
Murphy, Biham & Shamir first published 1990
powerful method to analyse block ciphers
used to analyse most current block ciphers with varying degrees of success
DES reasonably resistant to it, but Lucifer not
For DES it needs $2^{47}$ chosen plaintexts with its corresponding ciphertexts (impractical).

Differential Cryptanalysis

Assume to have a large number of $(x, x^*, y, y^*)$ using the same key, where $x' = x \oplus x^*$ is fixed. So this is chosen plaintext analysis.

Very similar to the linear cryptanalysis. Look into the last level first and try to find the key there. But now we will use property of xor of two inputs $x, x^*$, and others.

Property 1: For a fixed $x'$, the distribution of the xor of outputs from a S-box is quite non-uniform. (biased).

Define $\delta(x') = \{(x, x \oplus x')\}$ All the possible ordered pairs that $x \oplus x^* = x'$. How many elements in $\delta(1011)$? Must be $2^4$ if $x$ is a 4-bit string. Lets see the distribution of $N_d(x', y')= |\{(x, x^*) \in \delta(x') \& y \oplus y^* = y'\}|$
Biased Distribution for xor of outputs

<table>
<thead>
<tr>
<th>x</th>
<th>x'</th>
<th>y</th>
<th>y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1011</td>
<td>1110</td>
<td>1100</td>
</tr>
<tr>
<td>0001</td>
<td>1010</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>0010</td>
<td>1001</td>
<td>1101</td>
<td>1010</td>
</tr>
<tr>
<td>0011</td>
<td>1000</td>
<td>0001</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>1111</td>
<td>0010</td>
<td>0111</td>
</tr>
<tr>
<td>0101</td>
<td>1011</td>
<td>0110</td>
<td>1111</td>
</tr>
<tr>
<td>0110</td>
<td>1101</td>
<td>0111</td>
<td>1001</td>
</tr>
<tr>
<td>1000</td>
<td>1100</td>
<td>0010</td>
<td>1101</td>
</tr>
<tr>
<td>1001</td>
<td>0010</td>
<td>1010</td>
<td>1101</td>
</tr>
<tr>
<td>1010</td>
<td>0001</td>
<td>0110</td>
<td>0100</td>
</tr>
<tr>
<td>1011</td>
<td>0000</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>1100</td>
<td>0111</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>1101</td>
<td>0110</td>
<td>1001</td>
<td>1011</td>
</tr>
<tr>
<td>1110</td>
<td>0101</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>1111</td>
<td>0100</td>
<td>0111</td>
<td>0010</td>
</tr>
</tbody>
</table>

\[ x' = 1011 = x \text{ xor } x^* \]

See the y': very non-uniform
A uniform y' should give 1 for each item.
Define \( R_p(a', b') = \Pr(y' = b' \mid \text{ when } x' = a') \). For example:
\( R_p(1011, 0010) = 8/16 = \frac{1}{2} \)
How about other x'?
Xor not depend on key

Recall that the input to the $i$th S-box in round $r$ of the SPN from Example 3.1 is denoted $u^r_{(i)}$, and

$$u^r_{(i)} = w^{r-1}_{(i)} \oplus K^r_{(i)}.$$ 

An input xor is computed as

$$u^r_{(i)} \oplus (u^r_{(i)})^* = (w^{r-1}_{(i)} \oplus K^r_{(i)}) \oplus ((w^{r-1}_{(i)})^* \oplus K^r_{(i)})$$

Therefore, this input xor does not depend on the subkey bits used in round $r$; it

Property 2: the input xor does not depend on the key used in round $r$. So the given $x'$. It can pass through “key xor” stage.

Let’s see an example. If we know $w^3_3 \oplus w^3_3^*$, then we immediately know it is the same as $u^4_4 \oplus u^4_4^*$

An example

The arrow means xor = 1

$S_2^1$: $R_p(1011, 0010) = \frac{1}{2}$

$S_3^2$: $R_p(0100, 0110) = 3/8$

$S_2^3$: $R_p(0010, 0101) = 3/8$

$S_3^3$: $R_p(0010, 0101) = 3/8$

What is $\frac{1}{2} (3/8)^3 = 27/1024$?

$$x' = 0000 1011 0000 0000 \Rightarrow (x^3)' = 0000 0101 0101 0000$$

with probability $27/1024$. However,

$$(x^3)' = 0000 0101 0101 0000 \Rightarrow (x^4)' = 0000 0110 0000 0110.$$ 

Hence, it follows that

$$x' = 0000 1011 0000 0000 \Rightarrow (u^3)' = 0000 0110 0000 0110$$

with probability $27/1024$. Note that $(u^4)'$ is the xor of two inputs to the last round of S-boxes.

Select $(x, y) (x^*, y^*)$ pairs where $x \oplus x^* = x'$

For a guessed key, traverse back $y$ and $y^*$ to check if their $u \oplus u^* = u'$. Pick the the most likely key.
Block Cipher Design Principles

basic principles still like Feistel in 1970’s
number of rounds
  • more is better, exhaustive search best attack
function f:
  • provides “confusion”, is nonlinear, avalanche
key schedule
  • complex subkey creation, key avalanche

Modes of Operation

block ciphers encrypt fixed size blocks
eg. DES encrypts 64-bit blocks, with 56-bit key
need way to use in practise, given usually have arbitrary amount
  of information to encrypt
four were defined for DES in ANSI standard ANSI X3.106-1983
  Modes of Use
subsequently now have 5 for DES and AES
have block and stream modes
Electronic Codebook Book (ECB)

message is broken into independent blocks which are encrypted
each block is a value which is substituted, like a codebook, hence
name
each block is encoded independently of the other blocks
\[ C_i = \text{DES}_{K1} (P_i) \]
uses: secure transmission of single values
Advantages and Limitations of ECB

repetitions in message may show in ciphertext
- if aligned with message block
- particularly with data such graphics
- or with messages that change very little, which become a code-book analysis problem

weakness due to encrypted message blocks being independent
main use is sending a few blocks of data

Cipher Block Chaining (CBC)

message is broken into blocks
but these are linked together in the encryption operation
each previous cipher blocks is chained with current plaintext
  block, hence name
use Initial Vector (IV) to start process
  \[ C_i = \text{DES}_{K_1}(P_i \text{ XOR } C_{i-1}) \]
  \[ C_{-1} = \text{IV} \]
uses: bulk data encryption, authentication
Cipher Block Chaining (CBC)

Advantages and Limitations of CBC

each ciphertext block depends on **all** message blocks
thus a change in the message affects all ciphertext blocks after the change as well as the original block

need **Initial Value** (IV) known to sender & receiver
  - however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
  - hence either IV must be a fixed value (as in EFTPOS) or it must be sent encrypted in ECB mode before rest of message

at end of message, handle possible last short block
  - by padding either with known non-data value (eg nulls)
  - or pad last block with count of pad size
    - eg. [ b1 b2 b3 0 0 0 0 5 ] <- 3 data bytes, then 5 bytes pad+count

Dr. Edwin Sha
Cipher FeedBack (CFB)

message is treated as a stream of bits added to the output of the block cipher result is feedback for next stage (hence name) standard allows any number of bit (1, 8 or 64 or whatever) to be feedback

- denoted CFB-1, CFB-8, CFB-64 etc
is most efficient to use all 64 bits (CFB-64)

\[ C_i = P_i \oplus \text{DES}_{K_1}(C_{i-1}) \]

\[ C_{-1} = IV \]

uses: stream data encryption, authentication
Advantages and Limitations of CFB

appropriate when data arrives in bits/bytes
most common stream mode
limitation is need to stall while do block encryption after every n-bits
note that the block cipher is used in encryption mode at both ends
errors propagate for several blocks after the error

Output FeedBack (OFB)

message is treated as a stream of bits
output of cipher is added to message
output is then feed back (hence name)
feedback is independent of message
can be computed in advance
\[ C_i = P_i \text{ XOR } O_i \]
\[ O_i = \text{DES}_{K_1}(O_{i-1}) \]
\[ O_{-1} = \text{IV} \]
uses: stream encryption over noisy channels
Advantages and Limitations of OFB

used when error feedback a problem or where need to encryptions before message is available
superficially similar to CFB
but feedback is from the output of cipher and is independent of message
a variation of a Vernam cipher
• hence must never reuse the same sequence (key+IV)
sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs
originally specified with m-bit feedback in the standards
subsequent research has shown that only OFB-64 should ever be used
Counter (CTR)

a “new” mode, though proposed early on
similar to OFB but encrypts counter value rather than any feedback value
must have a different key & counter value for every plaintext block (never reused)
\[ C_i = P_i \oplus O_i \]
\[ O_i = DES_{K1}(i) \]
uses: high-speed network encryptions

Counter (CTR)

\[ \text{(a) Encryption} \]

\[ \text{(b) Decryption} \]
Advantages and Limitations of CTR

efficiency

- can do parallel encryptions
- in advance of need
- good for bursty high speed links
random access to encrypted data blocks
provable security (good as other modes)
but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Summary

have considered:
block cipher design principles
DES
- details
- strength
Differential & Linear Cryptanalysis
Modes of Operation
- ECB, CBC, CFB, OFB, CTR