IV. BLOCK CIPHERS

(a) Introduction

- symmetric key ciphers:
  \( \rightarrow \) same key used for encryption and decryption

- most classical ciphers operate on one character (~ few bits)
  \( \Rightarrow \) results in many of trivial attacks we have discussed

- one aspect to improving security and efficiency would be to have
  the encryption operation execute on blocks of plaintext bits
  (eg. 64 bit or 128 bit block)

  \( \rightarrow \) this is somewhat of a false generalization since one-time
  pad and its practical realizations (called stream ciphers)
  can encrypt one bit at a time

- best known block ciphers:

  Data Encryption Standard (DES)
  64-bit block with 56-bit key
  \( \rightarrow \) most applied cipher from late '70s to early 2000s

  Advanced Encryption Standard (AES)
  128-bit block with \( \geq 128 \)-bit key
  \( \rightarrow \) most applied cipher today

- Shannon (1949):

  \( \rightarrow \) ciphers can be constructed by taking product of 2 ciphers
  \( \Rightarrow \) if done carefully, product is more secure than base ciphers

  \( e(x) = e_{K2}(e_{K1}(x)) \)
- most block ciphers use Shannon's concept by iterating a number of rounds of simple cryptographic operations until enough security is obtained
  eg. simple Substitution-Permutation Network block cipher

(b) Substitution-Permutation Network (SPN)
- SPN is a simple model of a block cipher incorporating fundamentals of DES and AES

→ simple SPN is becoming popular in lightweight (i.e., low complexity) ciphers targeted to embedded systems such as RFID tags

- Shannon's principles:

  confusion
  → complex mathematical relationship (e.g., nonlinear) between plaintext and ciphertext

  diffusion
  → local effect in plaintext block spread across all ciphertext bits

Components of SPN

(1) S-box (substitution) ⇒ confusion

  S: \{0, 1\}^m \rightarrow \{0, 1\}^m

  → m-bit one-to-one mapping

  4×4 S-box mapping (i.e., \(m = 4\)) in hex:

<table>
<thead>
<tr>
<th>input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>E</td>
<td>4</td>
<td>D</td>
<td>1</td>
<td>2</td>
<td>F</td>
<td>B</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>3</td>
<td>A</td>
<td>6</td>
<td>C</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
(2) Permutation (transposition of bits) \Rightarrow \text{diffusion}

\[ \pi_p : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\} \quad \text{where } N \text{ is block size} \]

\[ \rightarrow \text{transposes bits between S-box output in one round to S-box input in next round} \]

16 bit permutation:

<table>
<thead>
<tr>
<th>input bit position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>output bit position</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input bit position</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>output bit position</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

(3) Key Scheduling

- each round of cipher must have subkey or round key applied

- subkeys derived from cipher key
  \[ \rightarrow \text{algorithm called key schedule} \]

- subkeys applied using XOR with cipher data

**General Comments**

- example 16-bit SPN is a "toy" cipher

  \[ \Rightarrow \text{using known plaintext attack could build a table of } 2^{16} \]
  ciphertext values indexed by plaintext values for a
cipher with a given key under attack
→ "dictionary attack"

⇒ practically secure block sizes are at least 64 bits

- SPN structure is simple and easy to implement in hardware and software

- can have same S-box mapping for every S-box in round (eg. AES) or different S-box mappings for each S-box in a round (eg. DES)

- AES is similar to SPN structure

→ 8×8 S-boxes, 128-bit block ⇒ 16 S-boxes per round

→ linear transformation used between S-box layers instead of permutations

→ 128-bit key version of AES uses 10 rounds

- in SPN structure, last round must be followed by subkey mixing or it would be trivial to remove last layer of S-boxes in an attack

- decryption is equivalent to going backwards through cipher and can be performed using same structure as encryption except:

  (1) inverse S-boxes must be used

  (2) subkeys applied in reverse order of rounds

  (3) placement of subkey bits must be moved around
(c) Linear Cryptanalysis

Reference: www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf

- takes advantage of probabilistic linear relationship between a subset of plaintext bits and subset of bits entering last round

→ resulting information can be used to determine consistency with last subkey candidates

⇒ most consistent subkey value gives correct subkey

→ for $R$-round cipher, after last round subkey found, last round can be removed and cipher attacked with $R - 1$ rounds

- linear expression to be exploited is of form

\[(*)\]

→ linear approximation of cipher

- let $p_L$ be defined as the probability that (*) holds

→ if $\{X_i\}, \{Y_j\}$ independent random variables that are equally likely to be 0 or 1, probability that (*) holds is $p_L = \frac{1}{2}$

→ LC exploits an expression like (*) for which $p_L \neq \frac{1}{2}$
\[ \rightarrow (*) \text{ is constructed by concatenating linear approximations of S-boxes and approximation of } p_L \text{ determined using piling-up lemma} \]

**Piling-Up Lemma**

- consider 2 independent binary random variables \( X_1 \) and \( X_2 \) where

- now

- define \( \epsilon_1 \) to be bias of \( X_1 \) \( \Rightarrow p_1 = \frac{1}{2} + \epsilon_1 \) and

\( \epsilon_2 \) to be bias of \( X_2 \) \( \Rightarrow p_2 = \frac{1}{2} + \epsilon_2 \)

leading to
- similarly

- in general, let $\varepsilon_1, 2, ..., k$ represent bias of random variable $X_1 \oplus X_2 \oplus \ldots \oplus X_k$ where $X_1, X_2, \ldots$ are all independent with biases $\varepsilon_1, \varepsilon_2, \ldots$

$$\varepsilon_{1, 2, \ldots, k} = \text{ Piling-Up Lemma}$$

- note that if $\varepsilon_i = 0$, for any $i$, (i.e., $p_i = \frac{1}{2}$)

$$\varepsilon_{1, 2, \ldots, k} = 0$$

\textit{Linear Approximations of S-boxes}

- to construct good linear approximations must examine S-boxes (only nonlinear component in SPN)

Consider SPN based on $4 \times 4$ S-box

with mapping from page 3
- from binary truth table of S-box can determine bias of any linear approximation for S-box

→ probability approximation holds

= number of rows where it holds / total number of rows

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$Y_1$</th>
<th>$Y_2$</th>
<th>$Y_3$</th>
<th>$Y_4$</th>
<th>$X_2 \oplus X_3 \oplus X_4$</th>
<th>$Y_1 \oplus Y_3 \oplus Y_4$</th>
<th>$X_1 \oplus X_4$</th>
<th>$Y_2$</th>
<th>$X_3 \oplus X_4$</th>
<th>$Y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

eg.
can combine all linear approximations in a "linear approximation table" using notation

\[ a \equiv (a_1, a_2, a_3, a_4) \text{ in hex} \]

\[ b \equiv (b_1, b_2, b_3, b_4) \text{ in hex} \]

where

\[(*)\]

represents linear approximation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>+8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>+6</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>-6</td>
<td>-2</td>
<td>-2</td>
<td>+2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-4</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+4</td>
<td>+2</td>
<td>-2</td>
<td>0</td>
<td>-4</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
<td>+4</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>+2</td>
<td>+4</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>-4</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+4</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>+2</td>
<td>+2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-4</td>
<td>0</td>
<td>-2</td>
<td>+2</td>
<td>0</td>
<td>+4</td>
<td>+2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>+4</td>
<td>-2</td>
<td>+2</td>
<td>-4</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>+4</td>
<td>0</td>
<td>-4</td>
<td>-4</td>
<td>0</td>
<td>+4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>-2</td>
<td>+4</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>+4</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
<td>-2</td>
<td>+4</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
<td>-4</td>
<td>-2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>0</td>
<td>+2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>+2</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>+4</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
- each element of table $E(a, b)$ is derived from count of number of times that (*) holds across all inputs (16 in total) minus half number of input values (8)

- bias is determined from $\varepsilon(a, b) = E(a, b) / 16$

eg. $a = 3, b = 9$

- LAT can be used to identify good linear approximation of S-boxes where $|\varepsilon(a, b)|$ is large

**Linear Approximation of Ciphers**

Idea: take good S-box approximations and connect them from round to round to derive a good $R - 1$ round linear approximation of cipher

$\rightarrow$ best approximation is the one with the largest absolute value of bias, i.e., $|\varepsilon_L|$ large or $p_L \gg \frac{1}{2}$ or $p_L \ll \frac{1}{2}$

- in determining $|\varepsilon_L|$ of overall cipher linear approximation, we assume S-box approximations are independent and use piling-up lemma

$$\varepsilon_L =$$

where $k$ is number of S-boxes in approximation and $\varepsilon_i$ represents approximation of "active" S-box $i$
\[ \Rightarrow |\varepsilon_L| \text{ maximized if} \]

(1) S-box \( |\varepsilon_i| \) maximized

(2) number of active S-boxes \((k)\) is minimized

Consider:
- assume SPN constructed using previously defined S-box

Let $U_j^i \equiv$ input bit $j$ to S-box in round $i$

$V_j^i \equiv$ output bit $j$ to S-box in round $i$

$K_j^i \equiv$ bit $j$ of subkey $i$

and using S-box approximations:

Round 1:

(1)

Round 2:

(2)

Round 3:

(3)

(4)

also

(5)

(6)

(7)

(8)

(9)

(10)
- now

\[(11)\]

\[(12)\]

\[(13)\]

\[(14)\]

- combining (1) to (14) by

\[(1) \oplus (2) \oplus (3) \oplus (4), \text{ then sub } U \text{ terms from (5-10) and } V^3 \text{ terms from (11-14)}\]

leads to

\[(**)\]

where
- approximation (**) has all intermediate bits canceled so that only bits from plaintext $P$ and input to last round $U^4$ are the only variables

eg. $(1) \oplus (2)$

→ key bits in $\Sigma K$ are fixed for cipher under attack and, hence,

(***)

with probability determined by Piling-Up Lemma, S-box approximation, and key values

- in fact,

$$\varepsilon_L = \begin{cases} \text{if } \Sigma K = 0 \\ \text{if } \Sigma K = 1 \end{cases}$$

But it is important to note that this analysis is based on the use of the Piling-Up Lemma which is based on assumption that each S-box approximation is independent of other S-box approximations (which is not strictly true!)
Recovering the Key

- can make use of (****) and fact that $|\varepsilon_L| = 1/32$ to recover subkey bits from $K^5$ associated with $S_{42}$ and $S_{44}$

- to proceed with attack as "known plaintext attack":

(1) collect a large number of plaintexts, $N_P$, and associated ciphertexts

(2) for each candidate partial subkey (i.e., 8 bits of $K^5$) partially decrypt last round of cipher and examine for consistency with

$$ (LA) $$

for all plaintext/ciphertext pairs

(3) increment a count for the candidate partial subkey whenever (LA) holds

(4) after all candidate partial subkeys have been considered, partial subkey with largest deviation from $N_P/2$ is assumed to be correct partial subkey

(5) try other linear approximations to get other subkey bits or when enough information is revealed, try exhaustive search on remaining cipher key bits
- attack is premised on assumption that (LA) (or its inverse) occurs
  significantly more often for correct partial subkey

  ⇒ incorrect candidate partial subkeys will generate relatively
  random \( U \) values and will have

  \[
  \text{count} \approx N_P/2
  \]

  ⇒ correct candidate partial subkey will have

  \[
  \text{count} \approx \]

- Matsui in original paper on linear cryptanalysis of DES shows
  that required \( N_P \) given by

  \[
  N_P \approx 
  \]

  to properly determine correct partial subkey with high
  probability where \( c \) is small constant

- for example \( \varepsilon^{-2} = 1024 \) and attack was experimentally successful
  for \( c \approx 8 \)

  \[
  \Rightarrow N_P \approx 8000 \text{ required in attack}
  \]
(d) **Differential Cryptanalysis**

- many similarities to linear cryptanalysis

  ⇒ both utilize information into last round to get bits from last subkey

- however, differential cryptanalysis is a *chosen* plaintext attack which uses plaintext pairs (and associated ciphertext) which satisfy a difference relationship

\[
(*)
\]

- if an appropriate $\Delta X$ is used, with high probability cryptanalyst can expect a particular difference for input to last round to occur:

- basic approach:

  1. select number of plaintext pairs $(X', X'')$ which satisfy (*)
(2) for last round, candidate partial subkeys (e.g. part of subkey $K^5$) partially decrypt ciphertext pairs and examine

(3) if $\Delta U^4$ is consistent with expectation, increment count for candidate partial subkey

(4) pick partial subkey with highest count to be correct

(5) determine other subkey bits similarly or use exhaustive search when appropriate

- concept works because incorrect partial subkey candidates result in random $\Delta U^4$ and expected $\Delta U^4$ no more likely to occur than others but correct partial subkey leads to frequent occurrence of expected $\Delta U^4$

- like linear cryptanalysis, S-box properties are examined to determine values for $\Delta X$ and $\Delta U^4$ that will occur with high probability

- see: [www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf](http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf) for thorough discussion of differential cryptanalysis applied to SPN
(e) Advanced Encryption Standard (AES)

- cipher Rijndael accepted as AES standard in 2001

  → will be the most significant cipher for years to come

- selected on basis of security (no successful attacks are known to exist) and efficiency (both in hardware and software implementations)

- basic concept:

  → SPN with permutation layer replaced with linear transformation where a linear transformation is defined such that each output bit of the operation is given by the XOR of a subset of input bits

- AES parameters:

  block size = 128 bits
  key size = 128, 192, 256 bits
  # rounds = 10, 12, 14 depending on key size
  S-boxes all use the same 8×8 mapping

- designed to be resistant to linear and differential attacks by

  (1) careful selection of S-box mapping

  (2) application of linear transformation instead of permutation

  ⇒ increases diffusion and forces more active S-boxes for both linear and differential cryptanalysis
Aside: Finite Fields

- finite fields appear frequently in cryptography

- informally, finite field is defined as a finite set of elements and operations addition and multiplication such that we can do addition and multiplication (and also subtract and divide) without leaving the set

- Note:

  additive inverse: \[ a + b = 0 \]

  \[ \Rightarrow b = -a \]

  i.e., \( b \) is inverse of \( a \)

  multiplicate inverse: \[ a \cdot b = 1 \]

  \[ \Rightarrow b = a^{-1} \]

  i.e., \( b \) is inverse of \( a \)

  subtraction: \[ a - b \equiv a + (-b) \]

  division: \[ a / b \equiv a \cdot b^{-1} \]

- in a finite field, all elements (except 0) must have additive and multiplicative inverse

eg. \( \mathbb{Z}_{11} = \{0, 1, 2, \ldots, 10\} \) with addition and multiplication modulo-11 is a finite field

for all \( a \in \mathbb{Z}_{11}, \quad -a = 11 - a \)

and \( a^{-1} \) exists because \( \gcd(a, 11) = 1 \)

(i.e., all elements of \( \mathbb{Z}_{11} \) are relatively prime to 11)
eg. \( \mathbb{Z}_{26} = \{0, 1, 2, \ldots, 25\} \) with addition and multiplication modulo-26 is not a finite field

for all \( a \in \mathbb{Z}_{26}, \quad -a = 26 - a \)

but \( a^{-1} \) does not necessarily exist

\( \rightarrow 13^{-1} \) is undefined

\( \Rightarrow \) cannot always divide

eg. \( 11/13 = ? \)

- as previously noted, \( a^{-1} \) can be determined using extended Euclidean algorithm (to come)

- in general, can easily define a finite field of order \( p \), where \( p \) is prime, which we represent as \( \text{GF}(p) \) after Galois

\[ \text{GF}(p): \quad p \text{ elements } \rightarrow \mathbb{Z}_p = \{0, 1, 2, \ldots, p - 1\} \text{ with } +, \times \text{ executed modulo-}p \]

- in computers, it is convenient to represent data in binary form

eg. \( n \) bits \( \Rightarrow 2^n \) different values

Can we define \( \text{GF}(2^n) \)? (or more generally \( \text{GF}(p^n) \)?)

eg. \( \text{GF}(2^3) \Rightarrow \) cannot use \( \mathbb{Z}_8 \) and modulo math since \( a^{-1} \) does not exist for all \( a \in \mathbb{Z}_8 \rightarrow \) not a finite field

- consider representing an \( n \)-bit vector as a polynomial

\[ \text{where} \quad \text{represents bits of vector} \]
addition/multiplication of polynomial operates on coefficients in GF(2) (i.e., addition is modulo-2)

eg.

GF(2^n): - represent 2^n n-bit vectors as polynomials
⇒ 2^n elements in the field
- +, × using modulo-2 math for co-efficients
- if polynomial has degree > n − 1, reduce using an irreducible polynomial \( m(x) \) of degree \( n \)
i.e., \( m(x) \) cannot be factored

eg. GF(2^3) using \( m(x) = x^3 + x + 1 \)

Note: additive and multiplicative inverses exist for all elements of GF(2^3)
Consider
Back to AES  (with figures from Stallings, 4th edition)

Structure:

128 bit block  → 4×4 array of bytes

11 128-bit round keys required → 44 round (sub) keys of 32 bits
Substitute Byte Transformation:

$\rightarrow$ see Table 5.4 in Stallings for S-box mapping

S-box:
- overall S-box can simply be implemented as table lookup or Boolean function

without regard to $(\cdot)^{-1}$ and $b(\cdot)$

eg $X = 95$

$\Rightarrow X^{-1} =$

$\Rightarrow Y =$

Why $(\cdot)^{-1}$? $\Rightarrow$ good cryptographic properties
  (eg. low biases for linear approximations)
Why $b(\cdot)$?  \[ \Rightarrow \text{does not affect properties of } (\cdot)^{-1} \text{ but ensures that math not all } \text{GF}(2^8) \]

Inverse S-box:

\[ \Rightarrow \text{properties similar to forward S-box} \]

*Shift Row Transformation:*

- a circular byte shift $\rightarrow$ row $i$ rotates $i - 1$ to left
Mix Column Transformation:

- mixes data within columns of state array $S$

so that

where

→ all math is done in $\text{GF}(2^8)$ using $m(x) = x^8 + x^4 + x^3 + x + 1$
eg.

- note only 2 multiplications needed (2× and 3×) and 3x = 2x + x

→ so can we simply implement 2×?  ⇒ YES!

Proof of (*):
Key Mixing:

- subkeys "added" to data using bit-by-bit XOR

\[
\begin{array}{ccccc}
  s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
  s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
  s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
  s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\
\end{array}
\]

\[ \oplus \]

\[
\begin{array}{ccccc}
  w_1 & w_{i+1} & w_{i+2} & w_{i+3} \\
\end{array}
\]

\[
= \begin{array}{cccc}
  s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\
  s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\
  s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\
  s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \\
\end{array}
\]

- for 128-bit key, takes 128 bits as first subkey and produces new subkey by using operations such as (1) rotation of bit positions, (2) substitution (using S-box) on bytes, and (3) bitwise XOR with a constant
Complete AES Encryption Round:
Decryption:

![Decryption Diagram]

Block Ciphers - 33
(f) Modes of Operation of Block Ciphers

- five basic modes of operation for block ciphers (others are also proposed)

*Electronic Codebook Mode (ECB)*

- straightforward application of block cipher

  \[ \rightarrow \text{given sequence of } n \text{ blocks, } n \text{ blocks of ciphertext} \]
  \[ \text{generated by encrypting each block separately under same key} \]

- decryption takes each ciphertext block \( y_i \) independently and produces plaintext block \( x_i \)

- problem:

  \[ y_i = y_j \quad \Rightarrow \quad x_i = x_j \]

  \[ \rightarrow \text{if attacker knows } (x_i, y_i) \text{ and then observes } y_j = y_i \]

  \[ \Rightarrow \text{plaintext } x_j \text{ revealed} \]

- *application*: secure transmission of single values (eg. encryption key during key distribution phase)
Cipher Block Chaining (CBC)

- typically used when encrypting large messages using block ciphers since it does not suffer from ECB problem

encryption

decryption

- applications:  (1) general purpose block encryption
                (2) authentication (to come)
Output Feedback Mode (OFB)

- block cipher produces keystream which is XORed with plaintext
  
  → keystream is a pseudorandom sequence of bits designed to mimic one-time pad

Let $z_i$ represent $m$ bit keystream character

→ keystream $z_1, z_2, \ldots, z_n$ produced by feeding back block cipher output to input

- initialization vector (IV) loaded into register $R$ to produce first ciphertext

- plaintext encrypted in groups of $m$ bits where $1 \leq m \leq N$ where $N$ is block size of cipher

- plaintext size can be any length in bits
  → does not need to be multiple of block length $N$
- larger $m \Rightarrow$ more efficient encryption

  eg. $m = 1 \rightarrow$ one block cipher operation required for every bit of ciphertext
      $m = N \rightarrow$ one block cipher operation required for every $N$ bits of ciphertext

- one bit in error in received plaintext (eg. due to noisy communications channel)
  $\Rightarrow$ one bit in error in recovered plaintext

- transmitter and receiver keystreams must be synchronized

  What if bit slip occurs? (eg. loss/insertion of bits due to timing error)

  $\Rightarrow$ decryption messed up and must be resynchronized by resending IV through signalling channel
  $\rightarrow$ lots of overhead

- block cipher decryption operation not needed

- application: stream-oriented transmission over noisy communication channels
**Cipher Feedback Mode (CFB)**

- ciphertext feeds back into keystream generator

- if slip or insertion occurs or if ciphertext block lost, CFB will
  \textit{self-synchronize}

- block cipher decryption operation not needed

- \textit{application}: general purpose stream-oriented transmission where
  slips or insertions can occur

- for $m$-bit $z_i$, we can recover from $m$-bit (or multiple $m$-bit )
  slip/insertion
  \[\rightarrow\] to recover from single bit slip/insertion must have $m = 1$
Counter Mode

- similar to OFB except uses a counter to drive block cipher input, rather than output feedback

- similar properties to OFB mode:
  1. 1 error in ciphertext $\Rightarrow$ 1 error in recovered plaintext
  2. synchronization of transmitter and receiver critical
  3. plaintext size can be any length in bits
     $\rightarrow$ does not need to be multiple of block length
  4. most efficient when $m = N$
  5. block cipher decryption operation not needed
Aside: Stream Ciphers

- another category of symmetric key ciphers
  → not as much confidence in practical cipher proposals as for block ciphers such as AES

- *stream ciphers* operate on one character at a time (e.g., one bit) and are designed to mimic a one-time pad by using a *keystream generator* to generate a pseudorandom sequence of bits that is difficult to predict, but which only requires a modest sized key of, say, 80 or 128 bits

- plaintext size can be any length in bits

- some block cipher modes designed to configure block cipher to operate as stream cipher (e.g., OFB, Counter)
**Error Propagation of Block Cipher Modes**

- different modes react differently to a bit error, a common occurrence in a noisy communications channel

- "error propagation" refers to the magnification effect caused by cipher system in response to a bit error in communications channel

<table>
<thead>
<tr>
<th>Mode</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>1 error in channel  ⇒ entire block corrupted</td>
</tr>
<tr>
<td>CBC</td>
<td>1 error in channel  ⇒ 1 block + 1 bit in next block corrupted</td>
</tr>
<tr>
<td>OFB</td>
<td>1 error in channel  ⇒ 1 bit error in recovered plaintext</td>
</tr>
<tr>
<td>Counter</td>
<td>1 error in channel  ⇒ 1 bit error in recovered plaintext</td>
</tr>
<tr>
<td>CFB</td>
<td>1 error in channel  ⇒ 1 bit error + entire next block corrupted</td>
</tr>
</tbody>
</table>

**Aside: Motivation for Public Key Cryptography**

- so far, we have only considered symmetric key ciphers, where encryption key = decryption key

But how do we get key $K$ at both ends of a communication if no secure channel exists?

Solution: *Asymmetric or Public Key Cryptography*