MODULE II

- Cryptography (i.e., Confusion and Diffusion)

**Big Idea #1: Confusion**

It's a good idea to obscure the relationship between your real message and your 'encrypted' message. An example of this 'confusion' is the trusty ol' Caesar Cipher:

**Plaintext:** ATTACK AT DAWN  
**Cipher:** DWWDFN DW GDZQ  
**A + 3 letters = D**

**Big Idea #2: Diffusion**

It's also a good idea to spread out the message. An example of this 'diffusion' is a simple column transposition:

**Plaintext:** ATTACK AT DAWN  
**Diffused:** ACD TKA TAW ATN  
**Diffused by 3 spots**
IDEA

• International Data Encryption Algorithm
• Symmetric block cipher
• 128 bit key
• Encrypt data in blocks of 64 bits
Cryptographic Strength

- Block Length
- Key Length
- Confusion
- Diffusion
IDEA Encryption

- Plaintext – 64 bits
- Key – 128 bits
- 8 rounds followed by a final transformation function
- Each of the rounds makes use of six 16 bit subkeys, where as the final transformation uses four subkeys, for a total of 52 subkeys
Diffusion

• Provided by the basic building block of algorithm known as Multiplication Addition Structure.
• Takes as inputs two 16 bit values derived from plaintext & two 16 bit subkeys derived from the key.
• Produces two 16-bit outputs
• This structure is repeated 8 times in algorithm
• Provides very effective diffusion
Multiplication Addition (MA) Structure
Details of a Single Round - Odd Round
Details of a Single Round- Odd Round (1)

- Round begins with a transformation
- That combines four input subblocks with four subkeys
- Using the addition & multiplication operations
- Four output blocks produce by this transformation are then combined using the XOR operation to form two 16 bit blocks that are input to the MA structure.
Details of a Single Round - Odd Round (2)

- MA structure also takes two subkeys as input
- Combines these inputs to produce two 16-bit outputs
- Finally the 4 output blocks from the upper transformation are combined with the two output blocks of MA structure using XOR to produce the 4 output blocks for this round.
- Second & third inputs are interchanged to produce the second & third output ($w_{12}$ & $w_{13}$)
Even Round

• Subsequent rounds have the same structure but with different subkey & plaintext derived inputs
Ninth stage – output
Transformation Stage
Ninth stage – output
Transformation Stage

• Second & third inputs are interchanged before being applied to the operational units
Subkey Generation

- 52, 16 bit subkeys are generated from 128-bit encryption key
- First eight subkeys, labeled Z₁,Z₂,...........Z₈ are taken directly from the key
  - Z₁ being equal to the first 16 bits
  - Z₂ being equal to the next 16 bits
- Then a circular left shift of 25 bit positions is applied to the key & the next 8 subkeys are extracted
- This procedure is repeated until all 52 subkeys are generated.
Single Round of IDEA (1\textsuperscript{st} Round)

Transformation

Sub-encryption
Table 2 for encryption sub key

| Round 1 | \( Z_{13} \) | \( Z_{14} \) | \( Z_{15} \) | \( Z_{16} \) | \( Z_{17} \) | \( Z_{18} \) | \( Z_{19} \) | \( Z_{20} \) | \( Z_{21} \) | \( Z_{22} \) | \( Z_{23} \) | \( Z_{24} \) | \( Z_{25} \) | \( Z_{26} \) | \( Z_{27} \) | \( Z_{28} \) | \( Z_{29} \) | \( Z_{30} \) |
|---------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| Round 2 | \( Z_{23} \) | \( Z_{24} \) | \( Z_{25} \) | \( Z_{26} \) | \( Z_{27} \) | \( Z_{28} \) | \( Z_{29} \) | \( Z_{30} \) | \( Z_{31} \) | \( Z_{32} \) | \( Z_{33} \) | \( Z_{34} \) | \( Z_{35} \) | \( Z_{36} \) | \( Z_{37} \) | \( Z_{38} \) | \( Z_{39} \) | \( Z_{40} \) |
| Round 3 | \( Z_{31} \) | \( Z_{32} \) | \( Z_{33} \) | \( Z_{34} \) | \( Z_{35} \) | \( Z_{36} \) | \( Z_{37} \) | \( Z_{38} \) | \( Z_{39} \) | \( Z_{40} \) | \( Z_{41} \) | \( Z_{42} \) | \( Z_{43} \) | \( Z_{44} \) | \( Z_{45} \) | \( Z_{46} \) | \( Z_{47} \) | \( Z_{48} \) |
| Round 4 | \( Z_{41} \) | \( Z_{42} \) | \( Z_{43} \) | \( Z_{44} \) | \( Z_{45} \) | \( Z_{46} \) | \( Z_{47} \) | \( Z_{48} \) | \( Z_{49} \) | \( Z_{50} \) | \( Z_{51} \) | \( Z_{52} \) | \( Z_{53} \) | \( Z_{54} \) | \( Z_{55} \) | \( Z_{56} \) | \( Z_{57} \) | \( Z_{58} \) |
| Round 5 | \( Z_{51} \) | \( Z_{52} \) | \( Z_{53} \) | \( Z_{54} \) | \( Z_{55} \) | \( Z_{56} \) | \( Z_{57} \) | \( Z_{58} \) | \( Z_{59} \) | \( Z_{60} \) | \( Z_{61} \) | \( Z_{62} \) | \( Z_{63} \) | \( Z_{64} \) | \( Z_{65} \) | \( Z_{66} \) | \( Z_{67} \) | \( Z_{68} \) |
| Round 6 | \( Z_{61} \) | \( Z_{62} \) | \( Z_{63} \) | \( Z_{64} \) | \( Z_{65} \) | \( Z_{66} \) | \( Z_{67} \) | \( Z_{68} \) | \( Z_{69} \) | \( Z_{70} \) | \( Z_{71} \) | \( Z_{72} \) | \( Z_{73} \) | \( Z_{74} \) | \( Z_{75} \) | \( Z_{76} \) | \( Z_{77} \) | \( Z_{78} \) |
| Round 7 | \( Z_{71} \) | \( Z_{72} \) | \( Z_{73} \) | \( Z_{74} \) | \( Z_{75} \) | \( Z_{76} \) | \( Z_{77} \) | \( Z_{78} \) | \( Z_{79} \) | \( Z_{80} \) | \( Z_{81} \) | \( Z_{82} \) | \( Z_{83} \) | \( Z_{84} \) | \( Z_{85} \) | \( Z_{86} \) | \( Z_{87} \) | \( Z_{88} \) |
| Round 8 | \( Z_{81} \) | \( Z_{82} \) | \( Z_{83} \) | \( Z_{84} \) | \( Z_{85} \) | \( Z_{86} \) | \( Z_{87} \) | \( Z_{88} \) | \( Z_{89} \) | \( Z_{90} \) | \( Z_{91} \) | \( Z_{92} \) | \( Z_{93} \) | \( Z_{94} \) | \( Z_{95} \) | \( Z_{96} \) | \( Z_{97} \) | \( Z_{98} \) |
| Round 9 | \( Z_{91} \) | \( Z_{92} \) | \( Z_{93} \) | \( Z_{94} \) | \( Z_{95} \) | \( Z_{96} \) | \( Z_{97} \) | \( Z_{98} \) | \( Z_{99} \) | \( Z_{100} \) | \( Z_{101} \) | \( Z_{102} \) | \( Z_{103} \) | \( Z_{104} \) | \( Z_{105} \) | \( Z_{106} \) | \( Z_{107} \) | \( Z_{108} \) |
IDEA Decryption

- Use the same structure (algorithm) as the encryption, but with different subkeys.
- Decryption subkeys $U_1, \ldots, U_{52}$ are derived from encryption subkeys.
AES

- Advanced Encryption Standard (AES)
- **designed by Rijndael**
- symmetric block cipher.
- plaintext block size of 128 bits, or 16 bytes.
- Key length can be 16, 24, or 32 bytes (128, 192, or 256 bits)
- The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.
AES

• Input to the encryption and decryption algorithms is a single 128-bit block.
• This block is depicted as a 4 * 4 square matrix of bytes.
• This block is copied into the State array, which is modified at each stage of encryption or decryption.
• After the final stage, State is copied to an output matrix.
Figure 3. State array input and output.
(a) Input, state array, and output.

(b) Key and expanded key.
Key & Expanded Key

• Similarly, the key is depicted as a square matrix of bytes.
• key is then expanded into an array of key schedule words.
• Each word is four bytes, and the total key schedule is 44 words for the 128-bit key.
• The ordering of bytes within a matrix is by column.
• So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix,
• the second four bytes occupy the second column, and so on.
AES Encryption Process

1. Initial transformation
2. Round 1
3. Round N - 1
4. Round N
5. Final state
6. Cipher text

Key expansion

<table>
<thead>
<tr>
<th>No. of rounds</th>
<th>Key Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>32</td>
</tr>
</tbody>
</table>
The first $N - 1$ rounds consist of four distinct transformation functions:

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

- The final round contains only three transformations
- Initial single transformation (AddRoundKey) before the first round, which can be considered as Round 0.
Encryption & Decryption

```
<table>
<thead>
<tr>
<th>Round #1</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext</td>
<td>w(0,3)</td>
</tr>
<tr>
<td>Add round key</td>
<td>Expand key</td>
</tr>
<tr>
<td>Substitutive bytes</td>
<td>w(4,7)</td>
</tr>
<tr>
<td>Shift row</td>
<td>Inverse sub bytes</td>
</tr>
<tr>
<td>Mix columns</td>
<td>Inverse shift row</td>
</tr>
<tr>
<td>Add round key</td>
<td>Inverse mix cols.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Round #9</th>
<th>Inverse mix cols.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add round key</td>
<td>Round #10</td>
</tr>
<tr>
<td>Substitutive bytes</td>
<td>Inverse sub bytes</td>
</tr>
<tr>
<td>Shift row</td>
<td>Inverse shift row</td>
</tr>
<tr>
<td>Mix columns</td>
<td>Inverse mix cols.</td>
</tr>
<tr>
<td>Add round key</td>
<td>Round #1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Round #10</th>
<th>Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add round key</td>
<td>Round #10</td>
</tr>
<tr>
<td>Substitutive bytes</td>
<td>Inverse sub bytes</td>
</tr>
<tr>
<td>Shift row</td>
<td>Inverse shift row</td>
</tr>
<tr>
<td>Add round key</td>
<td>Inverse mix cols.</td>
</tr>
</tbody>
</table>
```

Ciphertext
Overall AES structure (1)

- This structure is not a Feistel structure.
- In the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped.
- AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.
Overall AES structure (2)

- The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$.
- Four distinct words (128 bits) serve as a round key for each round.
- Four different stages are used, one of permutation and three of substitution:
Overall AES structure (3)

- **Substitute bytes**: Uses an S-box to perform a byte-by-byte substitution of the block
- **ShiftRows**: A simple permutation
- **MixColumns**: A substitution that makes use of arithmetic over GF(28)
- **AddRoundKey**: A simple bitwise XOR of the current block with a portion of the expanded key.
Overall AES structure (3)

• The structure is quite simple.
• For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.
Overall AES structure (3)

• Only the AddRoundKey stage makes use of the key.
• For this reason, the cipher begins and ends with an AddRoundKey stage.
• Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
Overall AES structure (3)

• The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable.

• The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key.
AES: PRIMITIVE OPERATIONS

- Substitute Bytes Transformation
- Forward and Inverse Transformations
- The forward substitute byte transformation, called SubBytes, is a simple table lookup as shown in Figure below.
Substitute Bytes

- a simple substitution of each byte
- uses one table of 16 x 16 bytes containing a permutation of all 256, 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- Galois Field- GF(p), where p is a prime number, is simply the ring of integers modulo p.
- designed to be resistant to all known attacks
Substitute Bytes
Shift Rows

- a circular byte shift in each
  - 1\textsuperscript{st} row is unchanged
  - 2\textsuperscript{nd} row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left

- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns
Shift Rows
Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in $\text{GF}(2^8)$ using prime poly $m(x) = x^8 + x^4 + x^3 + x + 1$

\[
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 \\
\end{bmatrix}
\begin{bmatrix}
S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\
S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\
S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\
S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\
\end{bmatrix}
= 
\begin{bmatrix}
S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\
S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\
S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\
S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\
\end{bmatrix}
\]
Mix Columns

\[
\begin{bmatrix}
2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2 \\
\end{bmatrix}
\times
\begin{bmatrix}
S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\
S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\
S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\
S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\
\end{bmatrix}
= 
\begin{bmatrix}
S_{0,0}' & S_{0,1}' & S_{0,2}' & S_{0,3}' \\
S_{1,0}' & S_{1,1}' & S_{1,2}' & S_{1,3}' \\
S_{2,0}' & S_{2,1}' & S_{2,2}' & S_{2,3}' \\
S_{3,0}' & S_{3,1}' & S_{3,2}' & S_{3,3}' \\
\end{bmatrix}
\]
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security
Add Round Key

\[
\begin{array}{cccc}
S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\
S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\
S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\
S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\
\end{array}
\oplus
\begin{array}{cccc}
W_i & W_{i+1} & W_{i+2} & W_{i+3} \\
\end{array}
= \begin{array}{cccc}
S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\
S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\
S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\
S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \\
\end{array}
\]
AES: Key Expansion

- AES key expansion algorithm takes as input a four-word (16-byte) key & produces a linear array of 44 words (176 bytes)
- The key is copied into the first four words of the expanded key.
- Remainder of the expanded key is filled in four words at a time.
- Each added word w[i] depends on the immediately preceding word, w[i - 1]
AES
Key Expansion

(a) Overall algorithm

(b) Function $g$

Figure 5.9 AES Key Expansion
Function g

1. RotWord performs a one-byte circular left shift on a word. This means that an input word \([B_0, B_1, B_2, B_3]\) is transformed into \([B_1, B_2, B_3, B_0]\).

2. SubWord performs a byte substitution on each byte of its input word, using the S-box.

3. The result of steps 1 and 2 is XORed with a round constant, \(R\text{con}[j]\).

• The round constant is a word in which the three right most bytes are always 0.
• Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
Encryption Round

• An encryption round has the structure
  SubBytes
  ShiftRows
  MixColumns
  AddRoundKey.
Decryption Round

• InvShiftRows
• InvSubBytes
• AddRoundKey
• InvMixColumns

• Thus, the first two stages of the decryption rounds need to be interchanged, and the second two stages of the decryption rounds need to be interchanged.
Interchanging InvShift Rows and InvSubBytes

- InvShiftRows affects the sequence of bytes in State but does not alter byte contents and does not depend on byte contents to perform its transformation.

- For a given State $Si$,

  $$\text{InvShiftRows} [\text{InvSubBytes} (Si)] = \text{InvSubBytes} [\text{InvShiftRows} (Si)]$$
Interchanging AddRoundKey and InvMixColumns

• The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State.

• If the key can be viewed as a sequence of words, then both AddRoundKey and InvMixColumns operate on State one column at a time.
Stream Ciphers

• process message bit by bit (as a stream)
• have a pseudo random **keystream**
• combined (XOR) with plaintext bit by bit
• randomness of **stream key** completely destroys statistically properties in message
  \[ C_i = M_i \text{ XOR StreamKey}_i \]
• but must never reuse stream key
  – otherwise can recover messages
Stream Cipher Structure

Key $K$

Pseudorandom byte generator (key stream generator)

$K$

 Plaintext byte stream $M$

Encryption

Ciphertext byte stream $C$

 Plaintext byte stream $M$

Pseudorandom byte generator (key stream generator)

$K$

$C$

$M$
Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity

- properly designed, can be as secure as a block cipher with same size key

- but usually simpler & faster
RC4

- Stream Cipher
- Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP/WPA)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input information processed a byte at a time
Schematic Representation of RC4
RC4 Key Schedule

- starts with an array S of numbers: 0..255
- A temporary vector T is also created
- If the length of key K is 256 bytes, then K is transferred to T
- For a key of length ‘keylen’ bytes, the first keylen elements of T are copied from K & then K is repeated as many times as necessary to fill out T.
Initialization

for $i = 0$ to $255$ do
    $S[i] = i$;
    $T[i] = K[i \mod \text{keylen}]$;

Initial permutation of $S$

$j = 0$;
for $i = 0$ to $255$ do
    $j = (j + S[i] + T[i]) \pmod{256}$
    swap $(S[i], S[j])$
RC4 Encryption (1)

- encryption continues shuffling array values
- **Stream Generation**
  
  ```
  i,j = 0;
  While (true)
    i = (i + 1) mod 256
    j = (j + S[i]) mod 256
    swap(S[i], S[j])
    t = (S[i] + S[j]) mod 256
    K = S[t];
  ```
RC4 Encryption (2)

• To encrypt, XOR the value k with the next byte of plain text.
• To decrypt, XOR the value k with the next byte of cipher text
RC4 Overview

(a) Initial state of S and T

(b) Initial permutation of S

(c) Stream Generation
RC4 Security

- claimed secure against known attacks
  have some analysis, none practical
- result is very non-linear
- since RC4 is a stream cipher, must never reuse a key
RC4

• Divided into 2 parts
  (i) Key Scheduling Algorithm (KSA)
  (ii) Pseudo Random Generation Algorithm (PRGA)
• Run PRGA on the KSA output to generate Key stream
• XOR the data with key stream