Chapter 5 – Advanced Encryption Standard

"It seems very simple."

"It is very simple. But if you don’t know what the key is it’s virtually indecipherable."

― Talking to Strange Men, Ruth Rendell

Origins

• clear a replacement for DES was needed
  — have theoretical attacks that can break it
  — have demonstrated exhaustive key search attacks
• can use Triple-DES – but slow, has small blocks
• US NIST issued call for ciphers in 1997
• 15 candidates accepted in Jun 98
• 5 were shortlisted in Aug-99
• Rijndael was selected as the AES in Oct-2000
• issued as FIPS PUB 197 standard in Nov-2001

The AES Cipher - Rijndael

• designed by Rijmen-Daemen in Belgium
• has 128/192/256 bit keys, 128 bit data
• an iterative rather than feistel cipher
  — processes data as block of 4 columns of 4 bytes
  — operates on entire data block in every round
• designed to be:
  — resistant against known attacks
  — speed and code compactness on many CPUs
  — design simplicity

AES Structure

➤ data block of 4 columns of 4 bytes is state
➤ key is expanded to array of words
➤ has 9/11/13 rounds in which state undergoes:
  ● byte substitution (1 S-box used on every byte)
  ● shift rows (permute bytes between groups/columns)
  ● mix columns (subs using matrix multiply of groups)
  ● add round key (XOR state with key material)
  ● view as alternating XOR key & scramble data bytes
➤ initial XOR key material & incomplete last round
➤ with fast XOR & table lookup implementation
### AES Structure

![AES Structure Diagram]

### Some Comments on AES

1. An **iterative** rather than Feistel cipher
2. Key expanded into array of 32-bit words
3. Four words form round key in each round
4. 4 different stages are used as shown
5. Has a simple structure
6. Only AddRoundKey uses key
7. AddRoundKey a form of Vernam cipher
8. Each stage is easily reversible
9. Decryption uses keys in reverse order
10. Decryption does recover plaintext

### Substitute Bytes

- A simple substitution of each byte
- Uses one table of 16x16 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)
- Designed to be resistant to all known attacks

### Substitute Bytes Example

- Example table for substitution bytes:

<table>
<thead>
<tr>
<th>1A</th>
<th>04</th>
<th>65</th>
<th>83</th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>65</td>
<td>70</td>
<td>96</td>
</tr>
<tr>
<td>5C</td>
<td>53</td>
<td>98</td>
<td>80</td>
</tr>
<tr>
<td>19</td>
<td>2D</td>
<td>AD</td>
<td>E5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>87</th>
<th>02</th>
<th>4D</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC</td>
<td>6D</td>
<td>4C</td>
<td>96</td>
</tr>
<tr>
<td>3A</td>
<td>C3</td>
<td>36</td>
<td>E7</td>
</tr>
<tr>
<td>9C</td>
<td>38</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

### Shift Rows

- A circular byte shift in each:
  - 1st row is unchanged
  - 2nd 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

### Substitute Bytes

- Example substitution:

  ![Substitute Bytes Example Diagram]
Shift Rows

- 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 GF(2^8)

Mix Columns

- uses arithmetic in the finite field GF(2^8)
- with irreducible polynomial
  \[ m(x) = x^8 + x^4 + x^3 + x + 1 \]
  which is \((100011011)\) or \{11b\}
- e.g.
  \[ (02) \cdot (87) \mod \{11b\} = (1 0000 1110) \mod \{11b\} = (1 0000 1110) \xor (1 0001 1011) = (0001 0101) \]

Mix Columns Example

\[
\begin{align*}
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 \\
\end{bmatrix} & \cdot \begin{bmatrix}
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\end{bmatrix} = \begin{bmatrix}
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\
\end{bmatrix} \\
\end{align*}
\]

AES Arithmetic

- can express each col as 4 equations
  – to derive each new byte in col
- decryption requires use of inverse matrix
  – with larger coefficients, hence a little harder
- have an alternate characterisation
  – each column a 4-term polynomial
  – with coefficients in GF(2^8)
  – and polynomials multiplied modulo \((x^4+1)\)
- coefficients based on linear code with maximal distance between codewords
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

AES Round

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU's
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description
AES Example

Key Expansion

<table>
<thead>
<tr>
<th>Word</th>
<th>0x00</th>
<th>0x01</th>
<th>0x02</th>
<th>0x03</th>
<th>0x04</th>
<th>0x05</th>
<th>0x06</th>
<th>0x07</th>
<th>0x08</th>
<th>0x09</th>
<th>0x0A</th>
<th>0x0B</th>
<th>0x0C</th>
<th>0x0D</th>
<th>0x0E</th>
<th>0x0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>x000</td>
<td>x001</td>
<td>x002</td>
<td>x003</td>
<td>x004</td>
<td>x005</td>
<td>x006</td>
<td>x007</td>
<td>x008</td>
<td>x009</td>
<td>x00A</td>
<td>x00B</td>
<td>x00C</td>
<td>x00D</td>
<td>x00E</td>
<td>x00F</td>
</tr>
<tr>
<td>W1</td>
<td>x010</td>
<td>x011</td>
<td>x012</td>
<td>x013</td>
<td>x014</td>
<td>x015</td>
<td>x016</td>
<td>x017</td>
<td>x018</td>
<td>x019</td>
<td>x01A</td>
<td>x01B</td>
<td>x01C</td>
<td>x01D</td>
<td>x01E</td>
<td>x01F</td>
</tr>
<tr>
<td>W2</td>
<td>x020</td>
<td>x021</td>
<td>x022</td>
<td>x023</td>
<td>x024</td>
<td>x025</td>
<td>x026</td>
<td>x027</td>
<td>x028</td>
<td>x029</td>
<td>x02A</td>
<td>x02B</td>
<td>x02C</td>
<td>x02D</td>
<td>x02E</td>
<td>x02F</td>
</tr>
<tr>
<td>W3</td>
<td>x030</td>
<td>x031</td>
<td>x032</td>
<td>x033</td>
<td>x034</td>
<td>x035</td>
<td>x036</td>
<td>x037</td>
<td>x038</td>
<td>x039</td>
<td>x03A</td>
<td>x03B</td>
<td>x03C</td>
<td>x03D</td>
<td>x03E</td>
<td>x03F</td>
</tr>
<tr>
<td>W4</td>
<td>x040</td>
<td>x041</td>
<td>x042</td>
<td>x043</td>
<td>x044</td>
<td>x045</td>
<td>x046</td>
<td>x047</td>
<td>x048</td>
<td>x049</td>
<td>x04A</td>
<td>x04B</td>
<td>x04C</td>
<td>x04D</td>
<td>x04E</td>
<td>x04F</td>
</tr>
<tr>
<td>W5</td>
<td>x050</td>
<td>x051</td>
<td>x052</td>
<td>x053</td>
<td>x054</td>
<td>x055</td>
<td>x056</td>
<td>x057</td>
<td>x058</td>
<td>x059</td>
<td>x05A</td>
<td>x05B</td>
<td>x05C</td>
<td>x05D</td>
<td>x05E</td>
<td>x05F</td>
</tr>
<tr>
<td>W6</td>
<td>x060</td>
<td>x061</td>
<td>x062</td>
<td>x063</td>
<td>x064</td>
<td>x065</td>
<td>x066</td>
<td>x067</td>
<td>x068</td>
<td>x069</td>
<td>x06A</td>
<td>x06B</td>
<td>x06C</td>
<td>x06D</td>
<td>x06E</td>
<td>x06F</td>
</tr>
<tr>
<td>W7</td>
<td>x070</td>
<td>x071</td>
<td>x072</td>
<td>x073</td>
<td>x074</td>
<td>x075</td>
<td>x076</td>
<td>x077</td>
<td>x078</td>
<td>x079</td>
<td>x07A</td>
<td>x07B</td>
<td>x07C</td>
<td>x07D</td>
<td>x07E</td>
<td>x07F</td>
</tr>
</tbody>
</table>

Encryption

AES Example

Avalanche

AES Decryption

• AES decryption is not identical to encryption since steps done in reverse
  • but can define an equivalent inverse cipher with steps as for encryption
    – but using inverses of each step
    – with a different key schedule
  • works since result is unchanged when
    – swap byte substitution & shift rows
    – swap mix columns & add (tweaked) round key

Implementation Aspects

• can efficiently implement on 8-bit CPU
  – byte substitution works on bytes using a table of 256 entries
  – shift rows is simple byte shift
  – add round key works on byte XOR’s
  – mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR’s
Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Summary

- have considered:
  - the AES selection process
  - the details of Rijndael – the AES cipher
  - looked at the steps in each round
  - the key expansion
  - implementation aspects