Public-key Cryptography
Theory and Practice

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Appendix A: Symmetric Techniques
A block cipher \( f \) of **block-size** \( n \) and **key-size** \( r \) is a function

\[
f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \rightarrow \mathbb{Z}_2^n
\]

that maps \((M, K)\) to \( C = f(M, K) \).

For each key \( K \), the map

\[
f_K : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^n
\]

taking a plaintext message \( M \) to the ciphertext message \( C = f_K(M) = f(M, K) \) should be bijective (invertible).

\( n \) and \( r \) should be large enough to preclude successful exhaustive search.

Each \( f_K \) should be a sufficiently random permutation.
## Block Ciphers: Examples

<table>
<thead>
<tr>
<th>Name</th>
<th>n, r</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES (Data Encryption Standard)</td>
<td>64, 56</td>
</tr>
<tr>
<td>FEAL (Fast Data Encipherment Algorithm)</td>
<td>64, 64</td>
</tr>
<tr>
<td>SAFER (Secure And Fast Encryption Routine)</td>
<td>64, 64</td>
</tr>
<tr>
<td>IDEA (International Data Encryption Algorithm)</td>
<td>64, 128</td>
</tr>
<tr>
<td>Blowfish</td>
<td>64, ⪅ 448</td>
</tr>
</tbody>
</table>

**The AES Finalists**

<table>
<thead>
<tr>
<th>Name</th>
<th>n, r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rijndael (Rijmen and Daemen)</td>
<td>128, 128/192/256</td>
</tr>
<tr>
<td>Serpent (Anderson, Biham and Knudsen)</td>
<td>128, 128/192/256</td>
</tr>
<tr>
<td>Twofish (Schneier and others)</td>
<td>128, ⪅ 256</td>
</tr>
<tr>
<td>RC6 (Rivest and others)</td>
<td>128, 128/192/256</td>
</tr>
<tr>
<td>MARS (Coppersmith and others)</td>
<td>128, 128–448 (multiple of 32)</td>
</tr>
</tbody>
</table>

**Old standard:** DES  
**New standard:** AES (adaptation of the Rijndael cipher)
Block Ciphers: Security Requirements

- **Introduced by Shannon in 1949.**

- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.

- **Diffusion**
  - The relation between plaintext and ciphertext must be very complex.
  - Changing a single plaintext bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each plaintext bit, each ciphertext bit should change with probability $1/2$.
  - Diffusion is meant to dissipate plaintext redundancy.
Iterated Block Cipher

(a) Encryption

Round 1
\[ M \xrightarrow{e} K_1 \xrightarrow{K_2} \cdots \xrightarrow{K_r} C \]

Round 2
\[ M \xrightarrow{e} K_2 \xrightarrow{K_1} \cdots \xrightarrow{K_r} C \]

Round \( l \)
\[ M \xrightarrow{e} K_{r-1} \xrightarrow{K_r} \cdots \xrightarrow{K_1} C \]

(b) Decryption

Round 1
\[ C \xleftarrow{e^{-1}} K_1 \xleftarrow{K_2} \cdots \xleftarrow{K_r} M \]

Round \( l \)
\[ C \xleftarrow{e^{-1}} K_r \xleftarrow{K_{r-1}} \cdots \xleftarrow{K_1} M \]
Substitution-Permutation Network (SPN)

Block from round \( i - 1 \)

Substitution Box (S–Box)

Permutation Box (P–Box)

\[ e \]

Round key \( K_i \)

Block to round \( i + 1 \)
Feistel Cipher

(a) Encryption

\[ L_i = R_{i-1} \]
\[ R_i = L_{i-1} \oplus e(R_{i-1}, K_i) \]

(b) Decryption

\[ R_{i-1} = L_i \]
\[ L_{i-1} = R_i \oplus e(L_i, K_i) \]
Proposed as a US standard in 1975.
- DES supports blocks of length $n = 64$ bits.
- DES supports keys $K = k_1 k_2 \ldots k_{64}$ of length $r = 64$ bits, but the bits $k_8, k_{16}, \ldots, k_{64}$ are used as parity-check bits. So the effective key size is 56 bits.
- DES is a Feistel cipher.
- The number of rounds in DES is $l = 16$. 
**DES Key Schedule**

**Input:** A DES key $K = k_1 k_2 \ldots k_{64}$.

**Output:** Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \text{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break $U_0$ in two 28-bit parts: $U_0 = C_0 \||| D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise}. \end{cases}$
  - Cyclically left shift $C_{i-1}$ by $s$ bits to get $C_i$.
  - Cyclically left shift $D_{i-1}$ by $s$ bits to get $D_i$.
  - Let $U_i := C_i \||| D_i = u_1 u_2 \ldots u_{56}$.
  - Compute 48-bit round key $K_i = \text{PC2}(U_i) = u_{14} u_{17} u_{11} \ldots u_{29} u_{32}$. 
## DES Key Schedule (contd)

<table>
<thead>
<tr>
<th></th>
<th>PC1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>58</td>
<td>50</td>
<td>42</td>
<td>34</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>19</td>
<td>11</td>
<td>3</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
</tr>
<tr>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>13</td>
<td>5</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PC2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>17</td>
<td>11</td>
<td>24</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>15</td>
<td>6</td>
<td>21</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>19</td>
<td>12</td>
<td>4</td>
<td>26</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>27</td>
<td>20</td>
<td>13</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>52</td>
<td>31</td>
<td>37</td>
<td>47</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>51</td>
<td>45</td>
<td>33</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>49</td>
<td>39</td>
<td>56</td>
<td>34</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>42</td>
<td>50</td>
<td>36</td>
<td>29</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
**Input:** Plaintext block \( M = m_1 m_2 \ldots m_{64} \) and round keys \( K_1, K_2, \ldots, K_{16} \).

**Output:** The ciphertext block \( C \).

- Apply initial permutation: \( V = IP(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_{7} \).
- Break \( V \) in two 32-bit parts: \( V = L_0 || R_0 \).
- For \( i = 1, 2, \ldots, 16 \), repeat the following steps:
  - \( L_i := R_{i-1} \).
  - \( R_i := L_{i-1} \oplus e(R_{i-1}, K_i) \).
- Let \( W = R_{16} || L_{16} = w_1 w_2 \ldots w_{64} \).
- Apply inverse of IP: \( C = IP^{-1}(W) = w_{40} w_8 w_{48} \ldots w_{57} w_{25} \).
### DES Encryption (contd)

#### IP

<table>
<thead>
<tr>
<th></th>
<th>58</th>
<th>50</th>
<th>42</th>
<th>34</th>
<th>26</th>
<th>18</th>
<th>10</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>56</td>
<td>48</td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>61</td>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
<td>21</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>63</td>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

#### IP$^{-1}$

<table>
<thead>
<tr>
<th></th>
<th>40</th>
<th>8</th>
<th>48</th>
<th>16</th>
<th>56</th>
<th>24</th>
<th>64</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>39</td>
<td>7</td>
<td>47</td>
<td>15</td>
<td>55</td>
<td>23</td>
<td>63</td>
<td>31</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>6</td>
<td>46</td>
<td>14</td>
<td>54</td>
<td>22</td>
<td>62</td>
<td>30</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>5</td>
<td>45</td>
<td>13</td>
<td>53</td>
<td>21</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>4</td>
<td>44</td>
<td>12</td>
<td>52</td>
<td>20</td>
<td>60</td>
<td>28</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>3</td>
<td>43</td>
<td>11</td>
<td>51</td>
<td>19</td>
<td>59</td>
<td>27</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>2</td>
<td>42</td>
<td>10</td>
<td>50</td>
<td>18</td>
<td>58</td>
<td>26</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>1</td>
<td>41</td>
<td>9</td>
<td>49</td>
<td>17</td>
<td>57</td>
<td>25</td>
</tr>
</tbody>
</table>
DES Encryption (contd)

Encryption primitive

\[ e(X, J) = P(S(E(X) \oplus J)), \]

where \( X \) is a 32-bit message block, and \( J \) is a 48-bit round key.

- Apply 32-to-48 bit expansion: \( X' = E(X) = x_{32}x_1x_2 \ldots x_{32}x_1 \).
- XOR with the round key: \( Y = X' \oplus J \).
- Break \( Y \) in eight 6-bit parts: \( Y = Y_1 \parallel Y_2 \parallel \cdots \parallel Y_8 \).
- For \( j = 1, 2, \ldots, 8 \), do the following:
  - Write \( Y_j = y_1y_2y_3y_4y_5y_6 \).
  - Consider the integers \( \mu = (y_1y_6)_2 \) and \( \nu = (y_2y_3y_4y_5)_2 \).
  - Let \( Z_j = (z_1z_2z_3z_4)_2 = S_j(\mu, \nu) \).
- Concatenate the \( Z_j \)'s to the 32-bit value: \( Z = Z_1 \parallel Z_2 \parallel \cdots \parallel Z_8 = z_1z_2 \ldots z_{32} \).
- Apply permutation function: \( e(X, J) = P(Z) = z_{16}z_7z_{20} \ldots z_4z_{25} \).
### DES Encryption (contd)

<table>
<thead>
<tr>
<th>E</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>26</td>
<td>19</td>
</tr>
<tr>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>31</td>
<td>11</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
</tr>
</tbody>
</table>

(Contd)
## DES Encryption: S-Boxes

### $S_1$

<table>
<thead>
<tr>
<th>14</th>
<th>4</th>
<th>13</th>
<th>1</th>
<th>2</th>
<th>15</th>
<th>11</th>
<th>8</th>
<th>3</th>
<th>10</th>
<th>6</th>
<th>12</th>
<th>5</th>
<th>9</th>
<th>0</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

### $S_2$

| 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
|----|---|---|----|---|----|---|---|---|---|---|----|----|---|---|---|---|
| 3  | 13| 4  | 7 | 15| 2  | 8 | 14| 12| 0  | 1 | 10 | 6 | 9 | 11| 5  |
| 0  | 14| 7  | 11| 10| 4  | 13| 1 | 5 | 8 | 12| 6  | 9 | 3 | 2 | 15 |
| 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11| 6 | 7 | 12 | 0 | 5 | 14| 9  |

### $S_3$

| 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
|----|---|---|----|---|---|----|---|---|----|----|---|----|---|---|---|---|
| 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10| 2 | 8 | 5 | 14 | 12| 11| 15| 1  |
| 13 | 6 | 4 | 9 | 8 | 15| 3 | 0 | 11| 1 | 2 | 12 | 5 | 10| 14| 7  |
| 1  | 10| 13| 0 | 6 | 9 | 8 | 7 | 4 | 15| 14| 3 | 11| 5 | 2 | 12 |
### DES Encryption: S-Boxes (contd)

#### $S_4$

<table>
<thead>
<tr>
<th>7</th>
<th>13</th>
<th>14</th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>9</th>
<th>10</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>5</th>
<th>11</th>
<th>12</th>
<th>4</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>12</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

#### $S_5$

| 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

#### $S_6$

| 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
### DES Encryption: S-Boxes (contd)

#### $S_7$

<table>
<thead>
<tr>
<th>4</th>
<th>11</th>
<th>2</th>
<th>14</th>
<th>15</th>
<th>0</th>
<th>8</th>
<th>13</th>
<th>3</th>
<th>12</th>
<th>9</th>
<th>7</th>
<th>5</th>
<th>10</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

#### $S_8$

| 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
|---|---|---|---|---|----|----|---|----|---|---|----|---|---|----|---|---|
| 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |
**Input:** Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

**Output:** The plaintext block $M$.

- Apply initial permutation: $V = \text{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break $V$ in two 32-bit parts: $V = R_{16} || L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
  - $R_{i-1} = L_i$.
  - $L_{i-1} = R_i \oplus e(L_i, K_i)$.
- Let $W = L_0 || R_0 = w_1 w_2 \ldots w_{64}$.
- Apply inverse of IP: $M = \text{IP}^{-1}(W) = w_{40} w_8 w_{48} \ldots w_{57} w_{25}$.

**Note:** DES decryption is the same as DES encryption, with the key schedule reversed.
AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.
- AES is not a Feistel cipher.
- The block size for AES is \( n = 128 \) bits.
- Number of rounds for AES is \( l = 10, 12, \) or 14 for key sizes \( r = 128, 192, \) or 256 bits.
- AES key schedule: From \( K \), generate 32-bit round keys \( K_0, K_1, \ldots, K_{4l+3} \). Four round keys are used in a round.
**State:** AES represents a 128-bit message block as a $4 \times 4$ array of octets:

Each octet $A = a_7a_6 \ldots a_1a_0$ in the state is identified with the element $a_7\alpha^7 + a_6\alpha^6 + \cdots + a_1\alpha + a_0$ of $F_{2^8} = F_2(\alpha)$, where $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$.

Each column $A_3A_2A_1A_0$ in the state is identified with the element $A_3y^3 + A_2y^2 + A_1y + A_0$ of $F_{2^8}[y]$ modulo the (reducible polynomial) $y^4 + 1$. 

<table>
<thead>
<tr>
<th>$\mu_0$</th>
<th>$\mu_4$</th>
<th>$\mu_8$</th>
<th>$\mu_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>$\mu_5$</td>
<td>$\mu_9$</td>
<td>$\mu_{13}$</td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$\mu_6$</td>
<td>$\mu_{10}$</td>
<td>$\mu_{14}$</td>
</tr>
<tr>
<td>$\mu_3$</td>
<td>$\mu_7$</td>
<td>$\mu_{11}$</td>
<td>$\mu_{15}$</td>
</tr>
</tbody>
</table>
AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key $K$.
- Convert the plaintext block $M$ to a state $S$.
- $S = \text{AddKey}(S, K_0, K_1, K_2, K_3)$. \[\text{bitwise XOR}\]
- for $i = 1, 2, \ldots, l$ do the following:
  - $S = \text{SubState}(S)$. \[\text{non-linear, involves inverses in } \mathbb{F}_{2^8}\]
  - $S = \text{ShiftRows}(S)$. \[\text{cyclic shift of octets in each row}\]
  - If $i \neq l$, $S = \text{MixCols}(S)$. \[\text{operation in } \mathbb{F}_{2^8}[y] \mod y^4 + 1\]
  - $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$. \[\text{bitwise XOR}\]
- Convert the state $S$ to the ciphertext block $C$. 
AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key $K$.
- Convert the ciphertext block $C$ to a state $S$.
- $S = \text{AddKey}(S, K_{4l}, K_{4l+1}, K_{4l+2}, K_{4l+3})$.
- For $i = l - 1, l - 2, \ldots, 1, 0$ do the following:
  - $S = \text{ShiftRows}^{-1}(S)$.
  - $S = \text{SubState}^{-1}(S)$.
  - $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
  - If $i \neq 0$, $S = \text{MixCols}^{-1}(S)$.
- Convert the state $S$ to the plaintext block $M$. 
AES: The AddKey Primitive

Let $S = (\sigma_{uv}) = \begin{bmatrix} 
\sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\
\sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\
\sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\
\sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} 
\end{bmatrix}$ be a state of AES.

Let the four 32-bit round keys be $L_0, L_1, L_2, L_3$ with the octet representation $L_u = \lambda_{u0}\lambda_{u1}\lambda_{u2}\lambda_{u3}$.

The $u$-th key $L_u$ is XORed with the $u$-th column of the state $S$.

$S$ maps to $\text{AddKey}(S, L_0, L_1, L_2, L_3) = \begin{bmatrix} 
\sigma_{00} \oplus \lambda_{00} & \sigma_{01} \oplus \lambda_{10} & \sigma_{02} \oplus \lambda_{20} & \sigma_{03} \oplus \lambda_{30} \\
\sigma_{10} \oplus \lambda_{01} & \sigma_{11} \oplus \lambda_{11} & \sigma_{12} \oplus \lambda_{21} & \sigma_{13} \oplus \lambda_{31} \\
\sigma_{20} \oplus \lambda_{02} & \sigma_{21} \oplus \lambda_{12} & \sigma_{22} \oplus \lambda_{22} & \sigma_{23} \oplus \lambda_{32} \\
\sigma_{30} \oplus \lambda_{03} & \sigma_{31} \oplus \lambda_{13} & \sigma_{32} \oplus \lambda_{23} & \sigma_{33} \oplus \lambda_{33} 
\end{bmatrix}$. 
AES: The SubState Primitive

Let $A = a_0a_1 \ldots a_6a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).

Let $B = b_0b_1 \ldots b_6b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).

Let $D = d_0d_1 \ldots d_6d_7 = 63 = 01100011$.

SubOctet($A$) $= C = c_0c_1 \ldots c_6c_7$, where

$c_i = b_i \oplus b_{(i+1) \text{rem} 8} \oplus b_{(i+2) \text{rem} 8} \oplus b_{(i+3) \text{rem} 8} \oplus b_{(i+4) \text{rem} 8} \oplus d_i$.

Let $S = \begin{array}{cccc}
\sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\
\sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\
\sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\
\sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33}
\end{array}$ be a state.

SubState($S$) $= \begin{array}{cccc}
\sigma'_{00} & \sigma'_{01} & \sigma'_{02} & \sigma'_{03} \\
\sigma'_{10} & \sigma'_{11} & \sigma'_{12} & \sigma'_{13} \\
\sigma'_{20} & \sigma'_{21} & \sigma'_{22} & \sigma'_{23} \\
\sigma'_{30} & \sigma'_{31} & \sigma'_{32} & \sigma'_{33}
\end{array}$, where $\sigma'_{uv} = \text{SubOctet}(\sigma_{uv})$. 
AES: The ShiftRows Primitive

Cyclically left rotate the $r$-th row by $r$ bytes:

\[
\begin{array}{cccc}
\sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\
\sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\
\sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\
\sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33}
\end{array}
\]

maps to

\[
\begin{array}{cccc}
\sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\
\sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{10} \\
\sigma_{22} & \sigma_{23} & \sigma_{20} & \sigma_{21} \\
\sigma_{33} & \sigma_{30} & \sigma_{31} & \sigma_{32}
\end{array}
\]
AES: The MixCols Primitive

Let $S = \begin{bmatrix} \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$ be a state.

Each column of $S$ is identified with an element of $\mathbb{F}_{2^8}[y]$, and is multiplied by the constant polynomial $[03]y^3 + [01]y^2 + [01]y + [02]$ modulo $y^4 + 1$.

The $\nu$-th column
\[
\begin{pmatrix} \sigma_{0\nu} \\ \sigma_{1\nu} \\ \sigma_{2\nu} \\ \sigma_{3\nu} \end{pmatrix}
\]
maps to
\[
\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \odot \begin{pmatrix} \sigma_{0\nu} \\ \sigma_{1\nu} \\ \sigma_{2\nu} \\ \sigma_{3\nu} \end{pmatrix},
\]

where $\odot$ is the multiplication of $\mathbb{F}_{2^8}$.
Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.
- \( \text{SubState}^{-1} \) is the octet-by-octet inverse of SubOctet.
- \( \text{SubOctet}^{-1} \) involves an affine transformation followed by taking inverse in \( \mathbb{F}_{2^8} \).
- \( \text{ShiftRows}^{-1} \) cyclically right rotates the \( r \)-th row by \( r \) bytes.
- \( \text{MixCols}^{-1} \) multiplies each column by the polynomial \([0b]y^3 + [0d]y^2 + [09]y + [0e]\) modulo \( y^4 + 1 \), with the coefficient arithmetic being that of \( \mathbb{F}_{2^8} \).
Let $t$ be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).

The respective numbers of rounds are $l = 10, 12, 14$.

AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key $K$.

Initially, $K = K_0 K_1 \ldots K_{t-1}$.

For $i = t, t + 1, \ldots, 4l + 3$, generate $K_i$ as follows:

- Let $K_{i-1} = \tau_0 \tau_1 \tau_2 \tau_3$ (each $\tau_j$ an octet).
- Let $\tau'_j = \text{SubOctet}(\tau_j)$.
- If ($i \equiv 0 \pmod{t}$), then set $T = (\tau'_1 \tau'_2 \tau'_3 \tau'_0) \oplus [\alpha^{(i/t)-1} || 000000]$, else if ($t > 6$) and ($i \equiv 4 \pmod{t}$), then set $T = \tau'_0 \tau'_1 \tau'_2 \tau'_3$.
- Generate the 32-bit key $K_i = K_{i-t} \oplus T$. 
Multiple Encryption

(a) Double encryption

(b) Triple encryption

\[ m \xrightarrow{K_1} g_1 \xrightarrow{x} g_2 \xrightarrow{K_2} c \]

\[ m \xrightarrow{K_1} h_1 \xrightarrow{x} h_2 \xrightarrow{y} h_3 \xrightarrow{K_3} c \]
Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leq n$.

To generate the ciphertext $C = C_1 C_2 \ldots C_l$.

**ECB (Electronic Code-Book) mode:** Here $n' = n$.
$$C_i = f_K(M_i).$$

**CBC (Cipher-Block Chaining) mode:** Here $n' = n$. Set $C_0 = IV$.
$$C_i = f_K(M_i \oplus C_{i-1}).$$

**CFB (Cipher FeedBack) Mode:** Here $n' \leq n$. Set $k_0 = IV$.
$$C_i = M_i \oplus \text{msb}_{n'}(f_K(k_{i-1})).$$
$$k_i = \text{lsb}_{n-n'}(k_{i-1}) \parallel C_i.$$ [Mask key and plaintext]

**OFB (Output FeedBack) Mode:** Here $n' \leq n$. Set $k_0 = IV$.
$$k_i = f_K(k_{i-1}).$$ [Generate next key]
$$C_i = M_i \oplus \text{msb}_{n'}(k_i).$$ [Mask plaintext block]

CFB and OFB modes act like stream ciphers.
Modes of Operation: Decryption

- **ECB (Electronic Code-Book) mode:**
  $M_i = f_K^{-1}(C_i)$.

- **CBC (Cipher-Block Chaining) mode:** Set $C_0 = IV$.
  $M_i = f_K^{-1}(C_i) \oplus C_{i-1}$.

- **CFB (Cipher FeedBack) Mode:** Set $k_0 = IV$.
  $M_i = C_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Remove mask from ciphertext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) || C_i$. [Generate next key]

- **OFB (Output FeedBack) Mode:** Set $k_0 = IV$.
  $k_i = f_K(k_{i-1})$. [Generate next key]
  $M_i = C_i \oplus \text{msb}_{n'}(k_i)$. [Remove mask from ciphertext]
Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
  - Exhaustive key search on block ciphers (like AES) with key sizes \( \geq 128 \) is infeasible.

- **Linear and differential cryptanalysis**
  - By far the most sophisticated attacks on block ciphers.
  - Impractical if sufficiently many rounds are used.
  - AES is robust against these attacks.
Attacks on Block Ciphers (contd)

- **Specific attacks on AES**
  - Square attack
  - Collision attack
  - Algebraic attacks (like XSL)

- **Meet-in-the-middle attack**
  - Applies to multiple encryption schemes.
  - For $m$ stages, we get security of $\lceil m/2 \rceil$ keys only.
Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: \( M = m_1 m_2 \ldots m_l \).
  Key stream: \( K = k_1 k_2 \ldots k_l \).
  Ciphertext stream: \( C = c_1 c_2 \ldots c_l \).
- **Encryption**: \( c_i = m_i \oplus k_i \).
- **Decryption**: \( m_i = c_i \oplus k_i \).
- Source of security: unpredictability in the key-stream.
- **Vernam’s one-time pad**: For a truly random key stream,
  \[
  \Pr(c_i = 0) = \Pr(c_i = 1) = \frac{1}{2}
  \]
  for each \( i \), irrespective of the probabilities of the values assumed by \( m_i \). This leads to **unconditional security**, that is, the knowledge of any number of plaintext-ciphertext bit pairs, does not help in decrypting a new ciphertext bit.
Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.
- It is difficult to generate truly random (and reproducible) key streams.
- Pseudorandom bit streams provide practical solution, but do not guarantee unconditional security.
- Pseudorandom bit generators are vulnerable to compromise of seeds.
- Repeated use of the same key stream degrades security.
Linear Feedback Shift Register (LFSR)

\[ D_{d-1} D_{d-2} \cdots D_2 D_1 D_0 \]

feedback

\[ s_d \]

\[ s_{d-1} \]

\[ s_{d-2} \]

\[ a_{d-1} \]

\[ a_{d-2} \]

output

\[ s_2 \]

\[ s_1 \]

\[ s_0 \]

\[ a_2 \]

\[ a_1 \]

\[ a_0 \]
## LFSR: Example

<table>
<thead>
<tr>
<th>Time</th>
<th>$D_3$</th>
<th>$D_2$</th>
<th>$D_1$</th>
<th>$D_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
LFSR: State Transition

- **Control bits**: $a_0, a_1, \ldots, a_{d-1}$.
- **State**: $s = (s_0, s_1, \ldots, s_{d-1})$.
- Each clock pulse changes the state as follows:

  $t_0 = s_1$
  $t_1 = s_2$
  \[ \vdots \]
  $t_{d-2} = s_{d-1}$
  $t_{d-1} \equiv a_0 s_0 + a_1 s_1 + a_2 s_2 + \cdots + a_{d-1} s_{d-1} \pmod{2}$.  

In the matrix notation $t \equiv \Delta_L s \pmod{2}$, where the transition matrix is

$$\Delta_L = \begin{pmatrix}
0 & 1 & 0 & \cdots & 0 & 0 \\
0 & 0 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 0 & 1 \\
 a_0 & a_1 & a_2 & \cdots & a_{d-2} & a_{d-1}
\end{pmatrix}.$$
LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.
- Maximum period of a non-zero bit-stream $= 2^d - 1$.
- **Maximum-length LFSR** has the maximum period.
- **Connection polynomial**

$$C_L(x) = 1 + a_{d-1}x + a_{d-2}x^2 + \cdots + a_1x^{d-1} + a_0x^d \in \mathbb{F}_2[X].$$

- $L$ is a maximum-length LFSR if and only if $C_L(x)$ is a primitive polynomial of $\mathbb{F}_2[x]$. 
An Attack on LFSR

- The linear relation of the feedback bit as a function of the current state in LFSRs invites attacks.

- **Berlekamp-Massey attack**
  Suppose that the bits $m_i$ and $c_i$ for $2d$ consecutive values of $i$ (say, 1, 2, ..., $2d$) are known to an attacker. Then $k_i = m_i \oplus c_i$ are also known for these values of $i$. Define the states $S_i = (k_i, k_{i+1}, \ldots, k_{i+d-1})$ of the LFSR. Then,

  $$S_{i+1} \equiv \Delta_L S_i \pmod{2}$$

  for $i = 1, 2, \ldots, d$. Treat each $S_i$ as a column vector. Then,

  $$\begin{pmatrix} S_2 & S_3 & \cdots & S_{d+1} \end{pmatrix} \equiv \Delta_L \begin{pmatrix} S_1 & S_2 & \cdots & S_d \end{pmatrix} \pmod{2}$$

  This reveals $\Delta_L$, that is, the secret $a_0, a_1, \ldots, a_{d-1}$.

- **Remedy**: Introduce non-linearity to the LFSR output.
A Nonlinear Combination Generator is shown in the diagram. It consists of multiple Linear Feedback Shift Registers (LFSRs) \( LFSR_1, \ LFSR_2, \ldots, \ LFSR_k \) connected together with a nonlinear function \( f \). The outputs of the LFSRs, \( u_1, u_2, \ldots, u_k \), are fed into the nonlinear function, resulting in an output \( u \).
The Geffe Generator

\[
\begin{align*}
\text{LFSR}_1 & \quad u_1 \\
\text{LFSR}_2 & \quad u_2 \\
\text{LFSR}_3 & \quad u_3 \\
\end{align*}
\]

\[k = f(u_1, u_2, u_3)\]
Nonlinear Filter Generator

Feedback function

Nonlinear filter function

Output
Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.
- Keyed hash functions authenticate source of messages.
- Symmetric techniques are typically used for designing hash functions.

- A **collision** for a hash function $H$ is a pair of two distinct strings $x, y$ with $H(x) = H(y)$.
- Since hash functions map an infinite domain to finite sets, collisions must exist for any hash function.
Hash Functions: Desirable Properties

- **Easy to compute**

- **First pre-image resistance (Difficult to invert):** For most hash values $y$, it should be difficult to find a string $x$ with $H(x) = y$.

- **Second pre-image resistance:** Given a string $x$, it should be difficult to find a different string $x'$ with $H(x') = H(x)$.

- **Collision resistance:** It should be difficult to find two distinct strings $x, x'$ with $H(x) = H(x')$. 
Collision resistance implies second pre-image resistance.

Second pre-image resistance does not imply collision resistance: Let $S$ be a finite set of size $\geq 2$ and $H$ a cryptographic hash function. Then

$$H'(x) = \begin{cases} 
0^{n+1} & \text{if } x \in S, \\
1 \|| H(x) & \text{otherwise,}
\end{cases}$$

is second pre-image resistant but not collision resistant.
Collision resistance does not imply first pre-image resistance: Let $H$ be an $n$-bit cryptographic hash function. Then

$$H''(x) = \begin{cases} 0 \| x & \text{if } |x| = n, \\ 1 \| H(x) & \text{otherwise}. \end{cases}$$

is collision resistant (so second pre-image resistant), but not first pre-image resistant.

First pre-image resistance does not imply second pre-image resistance: Let $m$ be a product of two unknown big primes. Define $H'''(x) = (1 \| x)^2 \pmod{m}$. $H'''$ is first pre-image resistant, but not second pre-image resistant.
Hash Functions: Construction

- **Compression function**: A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.

- **Merkle-Damgård’s meta method**
  - Break the input $x = x_1x_2 \ldots x_l$ to blocks each of bit-length $r$.
  - Initialize $h_0 = 0^r$.
  - For $i = 1, 2, \ldots, l$ use compression $h_i = F(h_{i-1} \parallel x_i)$.
  - Output $H(x) = h_l$ as the hash value.

![Diagram of Merkle-Damgård's meta method](attachment:image.png)
Hash Functions: Construction (contd)

- **Properties**
  - If $F$ is first pre-image resistant, then $H$ is also first pre-image resistant.
  - If $F$ is collision resistant, then $H$ is also collision resistant.

- **A concrete realization**
  Let $f$ is a block cipher of block-size $n$ and key-size $r$. Take:

  \[
  F(M || K) = f_K(M).
  \]

- **Keyed hash function**
  \[
  \text{HMAC}(M) = H(K || P || H(K || Q || M)),
  \]
  where $H$ is an unkeyed hash function, $K$ is a key and $P, Q$ are short padding strings.
The SHA (Secure Hash Algorithm) family:
SHA-1 (160-bit), SHA-256 (256-bit), SHA-384 (384-bit), SHA-512 (512-bit).

The MD family:
MD2 (128-bit), MD5 (128-bit).

The RIPEMD family:
RIPEMD-128 (128-bit), RIPEMD-160 (160-bit).
To compute SHA-1\( (M) \) for a message \( M \) of bit-length \( \lambda \).

- Pad \( M \) to generate \( M' = M \| 1 \| 0^k \| \Lambda \), where
  - \( \Lambda \) is the 64-bit representation of \( \lambda \), and
  - \( k \) is the smallest integer \( \geq 0 \) for which \( |M'| = \lambda + 1 + k + 64 \) is a multiple of 512.

- Break \( M' \) into 512-bit blocks \( M^{(1)}, M^{(2)}, \ldots, M^{(l)} \).

- Break each \( M^{(i)} = M_0^{(i)} \| M_1^{(i)} \| \cdots \| M_{15}^{(i)} \) into sixteen 32-bit words \( M_j^{(i)} \).
SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.
- Start with the initial hash value $H^{(0)} = 0x67452301\text{ efcdab89} 98badcfe 10325476\text{ c3d2e1f0}$.
- For $i = 1, 2, \ldots, l$, consume the message block $M^{(i)}$ to convert $H^{(i-1)}$ to $H^{(i)}$.
- Return $H^{(l)}$ as SHA-1$(M)$.
- Each $H^{(i)}$ is a 160-bit value.
- Write $H^{(i)} = H^{(i)}_0 \ || \ H^{(i)}_1 \ || \ H^{(i)}_2 \ || \ H^{(i)}_3 \ || \ H^{(i)}_4$, where each $H^{(i)}_j$ is a 32-bit word.
SHA-1: Compression Function

- Compute the message schedule $W_j$, $0 \leq j \leq 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
  - For $j = 16, 17, \ldots, 79$, set
    \[ W_j := LR^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16}) \]

- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in $t_j$.
- For $j = 0, 1, \ldots, 79$, do the following:
  - Set $T = \left( LR^5(t_0) + f_j(t_1, t_2, t_3) + t_4 + K_j + W_j \right) \text{ rem } 2^{32}$.
  - $t_4 = t_3$, $t_3 = t_2$, $t_2 = RR^2(t_1)$, $t_1 = t_0$, $t_0 = T$.
  - For $j = 0, 1, 2, 3, 4$, update $H_j^{(i)} := \left( t_j + H_j^{(i-1)} \right) \text{ rem } 2^{32}$. 
SHA-1: Compression Function (contd)

\[ f_j(x, y, z) = \begin{cases} 
xy \oplus \overline{x}z & \text{if } 0 \leq j \leq 19 \\
x \oplus y \oplus z & \text{if } 20 \leq j \leq 39 \\
xy \oplus xz \oplus yz & \text{if } 40 \leq j \leq 59 \\
x \oplus y \oplus z & \text{if } 60 \leq j \leq 79 
\end{cases} \]

\[ K_j = \begin{cases} 
0x5a827999 & \text{if } 0 \leq j \leq 19 \\
0x6ed9eba1 & \text{if } 20 \leq j \leq 39 \\
0x8f1bbcd0 & \text{if } 40 \leq j \leq 59 \\
0xca62c1d6 & \text{if } 60 \leq j \leq 79 
\end{cases} \]

LR^k and RR^k mean left and right rotate by \( k \) bits.
Attacks on Hash Functions

The **birthday attack** is based on the birthday paradox. For an $n$-bit hash function, one needs to compute on average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.

For cryptographic applications one requires $n \geq 128$ ($n \geq 160$ is preferable).

**Algebraic attacks** may make hash functions vulnerable.

Some other attacks:
- Pseudo-collision attacks
- Chaining attacks
- Attacks on the underlying cipher
- Exhaustive key search for keyed hash functions
- Long message attacks
Let $S$ be a set finite size $N$.

- $k$ elements are drawn at random from $S$ (with replacement).
- The probability that all these $k$ elements are distinct is

$$p_k = \frac{N(N-1) \cdots (N-k+1)}{N^k} = \prod_{i=1}^{k-1} \left(1 - \frac{i}{N}\right) \leq e^{\frac{-k(k-1)}{2N}}.$$  

- $p_k \leq 1/2$ for $k \geq \frac{1}{2} \sqrt{1 + 8N \ln 2} \approx 1.18 \sqrt{N}.$
- $p_k \leq 0.136$ for $k \geq 2 \sqrt{N}.$

**Examples**

- There is a chance of $\geq 50\%$ that at least two of $\geq 23$ (randomly chosen) persons have the same birthday.
- A collision of an $n$-bit hash function can be found with high probability from $O(2^{n/2})$ random hash calculations.