RECAP: HASH FUNCTIONS

- $H: D \rightarrow R$ where $D = \{0,1\}^*$ and $R=\{0,1\}^n$ for some integer $n > 0$

- Iterated Hash functions.

- Bit Padding Methods:
  - Method 1: Ambiguous.
MDC HASHING

- Single-length MDCs
  - Matyas-Meyer-Oseas
  - Davies-Meyer hash
  - Miyaguchi-Preneel hash

- Double-length MDCs:
  - MDC-2
  - MDC-4

- MD4 family
  - MD4, MD5
  - SHA-1
  - RIPEMD, RIPEMD-128, RIPEMD-160

- Modular Arithmetic Hash functions:
  - MASH, MASH-2

SINGLE-LENGTH MDCS

Common Definitions:

- \( E_k \): block cipher
  Symmetric-key Encryption functions.

- \( g \): maps n-bit input to a key \( K \) suitable for \( E_k \).
  - If \( K \) is n-bit length, \( g \) might be identity.

- \( IV \): initial value suitable for \( E_k \) usually of n-bit length. Fed at \( H_0 \).
MATYAS-MEYER-OSEAS HASH

Algorithm Matyas-Meyer-Oseas hash

INPUT: bitstring x.
OUTPUT: n-bit hash-code of x.

1. Input x is divided into n-bit blocks and padded, if necessary, to complete last block. Denote the padded message consisting of t n-bit blocks: \(x_1x_2\ldots x_t\). A constant n-bit initial value IV must be pre-specified.
2. The output is \(H_t\) defined by: \(H_0 = IV; H_i = E_{g(H_{i-1})}(x_i) \oplus H_{i-1}, 1 \leq i \leq t\).

DAVIES-MEYER HASH

Algorithm Davies-Meyer hash

INPUT: bitstring x.
OUTPUT: n-bit hash-code of x.

1. Input x is divided into k-bit blocks where k is the keysize. and padded, if necessary, to complete last block. Denote the padded message consisting of t k-bit blocks: \(x_1x_2\ldots x_t\). A constant n-bit initial value IV must be pre-specified.
2. The output is \(H_t\) defined by: \(H_0 = IV; H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}, 1 \leq i \leq t\).
**MIYAGUCHI-PRENEEL HASH**

$H_i$ defined by: $H_0 = IV; 
1 \leq i \leq t$

$H_i = E_{g(H_{i-1})}(x_i) \oplus x_i \oplus H_{i-1}$

**Algorithm Miyaguchi-Pieneel hash**

This scheme is identical to that of Algorithm 9.41, except the output $H_{i-1}$ from the previous stage is also XORed to that of the current stage. More precisely, $H_i$ is redefined as: $H_0 = IV; 
H_i = E_{g(H_{i-1})}(x_i) \oplus x_i \oplus H_{i-1}$, $1 \leq i \leq t$.

**TOGETHER: SINGLE-LENGTH MDCS!**

Matyas-Meyer-Oseas

Davies-Meyer

Miyaguchi-Pieneel
MDC HASHING

- Single-length MDCs
  - Matyas-Meyer-Oseas
  - Davies-Meyer hash
  - Miyaguchi-Preneel hash

- Double-length MDCs:
  - MDC-2
  - MDC-4

- MD4 family
  - MD4, MD5
  - SHA-1
  - RIPEMD, RIPEMD-128, RIPEMD-160

- Modular Arithmetic Hash functions:
  - MASH, MASH-2

DOUBLE-LENGTH MDCS


- Requires 2 (for MDC-2) and 4 (for MDC-4) Encryption operations.

- Encryption E is the DES encryption function.

- Padding might be necessary in all cases. Unambiguous is used!
**MDC-2 – BLOCK DIAGRAM!**

- \( E \): DES g, \( \hat{g} \): mapping function.
- X is multiple of 64 bits using unambiguous padding.
- A, B are the left and right halves (32 bits each), respectively.
- C, D are the left and right halves (32 bits each), respectively.

Notes in colored font are for illustration purposes only.

**MDC-2 ALGORITHM**

**Algorithm** MDC-2 hash function (DES-based)

INPUT: string \( x \) of bitlength \( r \geq 64t \) for \( t \geq 2 \).
OUTPUT: 128-bit hash-code of \( x \).

1. Partition \( x \) into 64-bit blocks \( x_i; x = x_1x_2 \ldots x_t \).
2. Choose the 64-bit non-secret constants \( IV, \tilde{IV} \) (the same constants must be used for MDC verification) from a set of recommended prescribed values. A default set of prescribed values is (in hexadecimal): \( IV = 0x5252525252525252, \tilde{IV} = 0x6b6b6b6b6b6b6b6b \).
3. Let \( \| \) denote concatenation, and \( C_i^L, C_i^R \) the left and right 32-bit halves of \( C_i \). The output is \( h(x) = H_t \| \tilde{H}_t \) defined as follows (for \( 1 \leq i \leq t \)):
   \[
   \begin{align*}
   H_0 &= IV; \\
   k_i &= g(H_{i-1}); \\
   C_i &= E_k(x_i) \oplus x_i; \\
   H_i &= C_i^L \| \tilde{C}_i^R; \\
   \tilde{H}_0 &= \tilde{IV}; \\
   \tilde{k}_i &= \tilde{g}(\tilde{H}_{i-1}); \\
   \tilde{C}_i &= E_{\tilde{k}_i}(x_i) \oplus x_i; \\
   \tilde{H}_i &= \tilde{C}_i^L \| C_i^R.
   \end{align*}
   \]
MDC-4 ALGORITHM

- Consists of two sequential execution of MDC-2.

---

X is multiple of 64 bits using unambiguous padding.

---

MDC-4 ALGORITHM

**Algorithm MDC-4 hash function (DES-based)**

INPUT: string $x$ of bitlength $r - 64t$ for $t \geq 2$. (See MDC-2 above regarding padding.)

OUTPUT: 128-bit hash-code of $x$.

1. As in step 1 of MDC-2 above.
2. As in step 2 of MDC-2 above.
3. With notation as in MDC-2, the output is $h(x) = G_t \mid \widetilde{G}_t$ defined as follows (for $1 \leq i \leq t$):

   - $G_0 = IV$; $\widetilde{G}_0 = IV$;
   - $k_i = \overline{g}(G_{i-1})$;
   - $C_i = E_{k_i} \binset x_i$;
   - $H_i = C_i \parallel \overline{C}_i$;
   - $\widetilde{k}_i = \overline{g}(\overline{G}_{i-1})$;
   - $\overline{C}_i = E_{k_i} \binset \overline{x}_i$;
   - $\overline{H}_i = \overline{C}_i \parallel C_i$;
   - $j_i = g(H_i)$;
   - $D_i = E_{j_i}(\overline{G}_{i-1}) \binset \overline{G}_{i-1}$;
   - $G_i = D_i \parallel \overline{D}_i$;
   - $\widetilde{j}_i = \overline{g}(\overline{H}_i)$;
   - $\overline{D}_i = E_{\overline{j}_i}(G_{i-1}) \binset \overline{G}_{i-1}$;
   - $\overline{G}_i = \overline{D}_i \parallel \overline{D}_i$.
MDC HASHING

- Single-length MDCs
  - Matyas-Meyer-Oseas
  - Davies-Meyer hash
  - Miyaguchi-Preneel hash
- Double-length MDCs:
  - MDC-2
  - MDC-4

MD4 family
  - MD4, MD5
  - SHA-1
  - RIPEMD, RIPEMD-128, RIPEMD-160
- Modular Arithmetic Hash functions:
  - MASH, MASH-2

MD4 FAMILY GENERAL IDEA!

- MD4: 128 bit hash code.
- MD5: strengthening of MD4 with 128-bit hash code.
- SHA-1: 160 bit hash code based on MD4.
- RIPEMD-160: 160 bit hash code based on MD4, MD5, and RIPEMD.
EXAMPLE HASH: MD4 FAMILY!

<table>
<thead>
<tr>
<th>Name</th>
<th>String</th>
<th>Hash value (as a hex string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>&quot;&quot;</td>
<td>31d6cfed169e913b73c55f6e74c689c0</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>bde52c31de33a4e6254e05f8d6f824</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>a48017aaf21d8f525fc10a87a8f729d</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>d79e1c308a5b8d6e8e06d3f412da9</td>
</tr>
<tr>
<td>MD5</td>
<td>&quot;&quot;</td>
<td>d41b52ef989e7a28e86ad55217bb189</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>0ce17f5c0b15a3e399e26972661</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>9004609c3d24b6266e9637f8d2e17f72</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>c3f3c376192a400707bbf39e6c67e</td>
</tr>
<tr>
<td>SHA-1</td>
<td>&quot;&quot;</td>
<td>da39a3ee5e6b4b0d1523f3b2f98b0a6</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>86f69e43f76a6dfae151d1b9b757517</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>a9999e3647066316a32e257785c26c9e0d6</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>32d16c7b8c986570c0a04e5c372a11b64</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>&quot;&quot;</td>
<td>9c1188a565c549c45d61089770eef5518b2255d31</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>0bdc942d56eb609dea54f44d835a467f6</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>8eb208c7665b0b44b0e8e98c6687715a0bfe</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>f71e27109c6092e1b596b9ceeb59b2665b3708d2c</td>
</tr>
</tbody>
</table>

RECAP: HASH FUNCTIONS: MDCS

- Non-Keyed Hash functions.
- Single-length MDCs
  - Matyas-Meyer-Oseas
  - Davies-Meyer hash
  - Miyaguchi-Pieneel hash
- Double-length MDCs:
  - MDC-2
  - MDC-4
- MD4 family
  - MD4, MD5
  - SHA-1
  - RIPEMD, RIPEMD-128, RIPEMD-160
- Modular Arithmetic Hash functions:
  - MASH, MASH-2

Grad level!
MESSAGE AUTHENTICATION CODES (MACS)

- Keyed Hash functions.
- Fewer than MDCs.
- Most popular is CBC-Based MACs.
- CBC-Based MACs is based on block ciphers such as DES.
- CRC-based MAC is based on stream ciphers.

CBC-Based MACs: Algorithm View!

When using DES as \( E \):
- \( N = 64 \) bits
- Key: 56 bits.
CBC-BASED MACS: ALGORITHM!

Algorithm CBC-MAC

INPUT: data $x$; specification of block cipher $E$; secret MAC key $k$ for $E$.
OUTPUT: $n$-bit MAC on $x$ ($n$ is the blocklength of $E$).

1. **Padding and blocking.** Pad $x$ if necessary (e.g., using Algorithm 9.30). Divide the padded text into $n$-bit blocks denoted $x_1, \ldots, x_t$.
2. **CBC processing.** Letting $E_k$ denote encryption using $E$ with key $k$, compute the block $H_i$ as follows: $H_1 \leftarrow E_k(x_1)$; $H_i \leftarrow E_k(H_{i-1} \oplus x_i)$, $2 \leq i \leq t$. (This is standard cipher-block-chaining. $IV = 0$, discarding ciphertext blocks $C_i = H_i$.)
3. **Optional process to increase strength of MAC.** Using a second secret key $k' \neq k$, optionally compute: $H'_i \leftarrow E^{-1}_{k'}(H_i)$, $H_i \leftarrow E_k(H'_i)$. (This amounts to using two-key triple-encryption on the last block; see Remark 9.59.)
4. **Completion.** The MAC is the $n$-bit block $H_t$.

- Compulsory reading sections 9.1 – 9.5 for Hashing!
- End lecture 10.
- References: