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 $H_0$.  

![Diagram of Matyas-Meyer-Oseas hash function]
**MATYAS-MEYER-OSEAS HASH**

$H_0$ defined by: $H_0 = IV$;

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

$g(H_{i-1})$ is the Key

**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: $x_1, x_2, \ldots x_t$. A constant $n$-bit initial value $IV$ must be pre-specified.

2. The output is $H_t$ defined by: $H_0 = IV; H_i = E_{g(H_{i-1})}(x_i) \oplus x_i$, $1 \leq i \leq t$.

---

**DAVIES-MEYER HASH**

$H_0 = IV; 1 \leq i \leq t$.

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

$x_i$ is the Key

**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: $x_1, x_2, \ldots x_t$. 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-Preneel 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-Preneel
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, ĝ: 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 $r = 64t$ for $t \geq 2$

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

1. Partition $x$ into 64-bit blocks $x_i$: $x = x_1 x_2 \ldots x_t$.  
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): $IV = 0x5252525252525252$. $\tilde{IV} = 0x2525252525252525$.  
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 $1 \leq i \leq t$):
   
   $$
   \begin{align*}
   H_0 &= IV; & k_1 &= g(H_{i-1}); & C_i &= E_k(x_i) \oplus x_i; & H_i &= C_i^L \| \tilde{C}_i^R \\
   \tilde{H}_0 &= \tilde{IV}; & \tilde{k}_i &= \tilde{g}(H_{i-1}); & \tilde{C}_i &= E_{\tilde{k}_i}(x_i) \oplus x_i; & \tilde{H}_i &= \tilde{C}_i^L \| \tilde{C}_i^R.
   \end{align*}
   $$
   $$
MDC-4 Algorithm

- Consists of two sequential execution of MDC-2.

X is multiple of 64 bits using unambiguous padding.

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 $h(x) = G_t \parallel \tilde{G}_t$ defined as follows (for $1 \leq i \leq t$):

$$ G_0 = IV; \quad \tilde{G}_0 = \tilde{IV}; $$

$$ k_i = g(G_{i-1}); \quad C_i = E_{k_i}(x_i) \oplus x_i; \quad H_i = C_i^L \parallel \tilde{C}_i^R $$

$$ \tilde{k}_i = \tilde{g}(G_{i-1}); \quad \tilde{C}_i = E_{\tilde{k}_i}(x_i) \oplus x_i; \quad \tilde{H}_i = \tilde{C}_i^L \parallel C_i^R $$

$$ j_i = g(H_i); \quad D_i = E_{j_i}(G_{i-1}) \oplus \tilde{G}_{i-1}; \quad G_i = D_i^L \parallel \tilde{D}_i^R $$

$$ \tilde{j}_i = \tilde{g}(\tilde{H}_i); \quad \tilde{D}_i = E_{\tilde{j}_i}(G_{i-1}) \oplus \tilde{G}_{i-1}; \quad \tilde{G}_i = \tilde{D}_i^L \parallel D_i^R. $$
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 byte string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>&quot;&quot;</td>
<td>31d6cfe0dd8ae931b73c59d7e0c0c89cc0</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>bde52cb31e33e4626e5dbf0dbf24</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>a48017aaf21d5f25fcbf0ae8aa6729d</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>d79e1c308a5b8bdeea8e6e36f412da0</td>
</tr>
<tr>
<td>MD5</td>
<td>&quot;a&quot;</td>
<td>d41d8cd98909866f00e9800998ecf8427e</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>0cc175b9c2f1bda8831c3394e2677259d</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>900156964c0e10af00e8d9661b2e1c77272</td>
</tr>
<tr>
<td>SHA-1</td>
<td>&quot;a&quot;</td>
<td>da39a5eeb78d92f4a62e9d0df1b7da41f5</td>
</tr>
<tr>
<td></td>
<td>&quot;abc&quot;</td>
<td>8664c6237d0294283a5f2f30f49f21d0</td>
</tr>
<tr>
<td></td>
<td>&quot;abcdefghijklmnopqrstuvwxyz&quot;</td>
<td>a9999e2e657ae8119a6b9c0df5b6908d0</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>&quot;a&quot;</td>
<td>9c11b93c5f5c54a6247b9e811b307dd130da968c60c33b9e88f4c5802b6b40d197f1b7879fc98f60f00d0b68f71c27109ce692ebad0e1b5b6704d8d865b3708600f0</td>
</tr>
</tbody>
</table>

**Recap: Hash Functions: MDCs**

- Non-Keyed Hash functions.
- 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

**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: $H_i' \leftarrow E_k^{-1}(H_i), H_i \leftarrow E_k(H_i')$. (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 11.
- References: