Hashes and Message Digests (Chapter 5)

Outline

- Introduction to Hash
- MD2
- MD5
- SHA-1

Hashes

- Hash is also called message digest
- One-way function: \( d=h(m) \) but no \( h'(d)=m \)
  - Cannot find the message given a digest
- Cannot find \( m_1, m_2 \), where \( d_1=d_2 \)
- Arbitary-length message mapped to fixed-length digest
- Randomness
  - any bit in the outputs: ‘1’ half of the time
  - each output: 50% ‘1’ bits

How Many Bits for Hash?

- Many messages yield the same digest
  - E.g., 1000-bits message, 128-bits digest
- \( m \) bits hashes, takes about \( 2^{m/2} \) to find two with the same hash (>= 50% prob.)
- 64 bits, takes \( 2^{32} \) messages to search (doable)
- Need at least 128 bits, \( 2^{64} \) search
Birthday Problem

- Compute probability of different birthdays
- Random sample of $n$ people (birthdays) taken from $k$ (365) days
- There are $n(n-1)/2$ pairs of input
- Prob. of $1/k$ of both inputs maps to the same output (not a particular one though)
- $k/2$ pairs are needed for the prob. of 50% to find the matching pair
- $n$ greater than $\sqrt{k}$ will do.

Using Hash for Authentication

- Alice to Bob: challenge $r_A$ (e.g., “I am Alice”)
- Bob to Alice: $\text{MD}(K_{AB} \| r_A)$
- Bob to Alice: $r_B$ (e.g., “I am Bob”)
- Alice to Bob: $\text{MD}(K_{AB} \| r_B)$
- Only need to compare MD results

Using Hash to Compute MAC: integrity

- Cannot just compute $\text{MD}(m)$
- Keyed Hash:
  - MAC: $\text{MD}(K_{AB} \| m)$, almost work, however…
    - Allows concatenation with additional message: $\text{MD}(K_{AB} \| m')$
    - MD through chunk $n$ depends on MD through chunks $n-1$ and the data in chunk $n$
  - Proposal 1: Put secret at the end of message:
    - MD ($m \| K_{AB}$)
  - Proposal 2: HMAC
    - MD ($K_{AB} \| \text{MD}(K_{AB} \| m)$)

Using Hash to Encrypt: confidentiality

- Challenge: Decryption
  - One-time pad: similar to OFB
    - compute pseudorandom bit streams using MD, $K$, and IV
      - $b_i=\text{MD}(K_{AB} \| \text{IV}), b_i=\text{MD}(K_{AB} \| p_{i-1}), \ldots$
      - $\oplus$ with message blocks
  - Or mixing in the plaintext
    - similar to cipher feedback mode (CFB)
      - $b_i=\text{MD}(K_{AB} \| \text{IV}), c_i=p_i \oplus b_1$
      - $b_2=\text{MD}(K_{AB} \| c_1), c_2=p_2 \oplus b_2$
    - Decryption w/ the same order as encryption
Output Feedback Mode (OFB)

IV (n bits) Like a Random Number Generator...

\[
\begin{align*}
M_1 &\rightarrow C_1 \\
M_2 &\rightarrow C_2 \\
M_3 &\rightarrow C_3 \\
M_4 &\rightarrow C_4
\end{align*}
\]

⊕ message w/ as many bits of pseudo-random stream as necessary

OFB is a stream cipher

General k-bit Cipher Feedback Mode (CFB)

\[
\begin{align*}
IV &\quad k \\
M_1 &\rightarrow ENC &\quad M_2 &\rightarrow ENC &\quad M_3 &\rightarrow ENC &\quad M_4 &\rightarrow ENC
\end{align*}
\]

\[
\begin{align*}
K &\quad \text{bits} &\quad K &\quad \text{bits} &\quad K &\quad \text{bits}
\end{align*}
\]

MD2

- 128-bit message digest:
  - Arbitrary number of bytes of message
  - First pad to multiple of 16 bytes
  - Append MD2 checksum (16 bytes) to the end
    - The checksum is almost a MD, but not cryptographically secure by itself.
  - Final pass: process whole message, 16 bytes at a time

MD2 Padding

\[
\begin{align*}
\text{original message} &\rightarrow r \text{ octets (1 \leq r \leq 16)} \\
&\quad \text{each containing } r \\
&\quad \text{multiple of 16 octets}
\end{align*}
\]

Figure 5-3. MD2 Padded Message

- If the message is multiple of 16 octets, 16 bytes to be added
- Otherwise, 1~15 bytes to be added
MD2 Checksum (start w/ 0)

- One byte at a time, \( k \times 16 \) steps (in total)
- \( m_{nk} \): \( nk^{th} \) byte of message, \( n: 1-16 \)
- \( c_n = \pi(m_{nk} \oplus c_{n-1}) \oplus c_n \)
- \( \pi: 0 \rightarrow 41, 1 \rightarrow 46, \ldots \)
  - Substitution on 0-255 (value of the byte)

MD2 Final Process

- Operate on 16-byte chunks
- 48-byte quantity \( q \):
  - (current digest | message chunk | digest \( \oplus \) chunk)
- 18 passes of massaging over \( q \), and one byte at a time:
  - \( c_n = c_{n-1} \oplus c_n \) for \( n = 0, \ldots, 47 \); \( c_{-1} = 0 \) for pass 0.
  - After pass 17, use first 16 bytes as new digest
  - After \( 16 \times 8 = 128 \)
- Repeat till the entire padded message is processed

MD5: Message Digest Version 5

- Input Message
- Output 128 bits Digest

Overview of MD5

- More "Conservative" than MD4: Less concerned w/ speed, more concerned w/ security
### MD 5 Padding

- Given original message $M$, add padding bits “10*” such that resulting length is 64 bits less than a multiple of 512 bits.
- Append *(original length in bits) mod 2^64*, represented in 64 bits to the padded message.
- Final message is chopped 512 bits a block.

### MD5: Padding

- Input Message
- 512 bit block
- Padding
- Initial Value
- Transformation block by block
- Output 128 bits Digest
- Final Output

### MD5 Blocks

- 512: $B_1$
- 512: $B_2$
- 512: $B_3$
- 512: $B_4$
- Result

### MD5 Box

- 512-bit message chunks (16 words)
- Initial 128-bit vector
- 128-bit result

**F:** $(x \land y) \lor (\neg x \land z)$

**G:** $(x \land z) \lor (y \land \neg z)$

**H:** $x \oplus y \oplus z$

**I:** $y \oplus (x \land \neg z)$

**+:** binary sum

$x \ll y$: $x$ left rotate $y$ bits
**MD5 Process**

- As many stages as the number of 512-bit blocks in the final padded message
- Digest: 4 32-bit words: MD=A|B|C|D
- Every message block contains 16 32-bit words: \( m_0|m_1|m_2|...|m_{15} \)
  - Digest MD\(_i\) initialized to:
    - A=01234567, B=89abcdef, C=fedcba98, D=76543210
  - Every stage consists of 4 passes over the message block, each modifying MD; each pass involves different operation

**Different Passes...**

- Different functions and constants are used
- Different set of \( m_i \) is used, \( i=0, 1, ...15 \)
  - Each pass goes through all 16 message words (each is 32-bit)
- Different set of shift amount is used

**Functions and Random Numbers**

- \( F(x,y,z) = (x \land y) \lor (~x \land z) \)
- \( G(x,y,z) = (x \land z) \lor (y \land ~z) \)
- \( H(x,y,z) = x \oplus y \oplus z \)
- \( I(x,y,z) = y \oplus (x \land ~z) \)
- \( T_i = \text{int}(2^{32} \times \text{abs(sin}(i))), 0<i<65 \)

**Processing of Block \( m_i \) - 4 Passes**

- \( m_i \)
- \( AB = f_\theta(ABCD, m_i, T[1..16]) \)
- \( BC = f_\theta(ABCD, m_i, T[17..32]) \)
- \( CD = f_\theta(ABCD, m_i, T[33..48]) \)
- \( DE = f_\theta(ABCD, m_i, T[49..64]) \)
- \( MD_{i+1} \)

Ref: p138-p139

Convention:
- \( A = d_0 \); \( B = d_1 \)
- \( C = d_2 \); \( E = d_3 \)
- \( T_i \): diff. constant
**Secure Hash Algorithm (SHA)**

- Developed by NIST, specified in the Secure Hash Standard (SHS, FIPS Pub 180-3), 1993
- SHA is specified as the hash algorithm in the Digital Signature Standard (DSS), NIST

**General Logic**

- Input message must be < 2^{64} bits
  - not really a problem
- Message is processed in 512-bit blocks sequentially
- Message digest is 160 bits
- SHA design is similar to MD5, but a lot stronger

**Basic Steps**

**Step 1:** Padding
**Step 2:** Appending length as 64 bit unsigned
**Step 3:** Initialize MD buffer 5 32-bit words

A|B|C|D|E  
A = 67452301  
B = efcdab89  
C = 98badcfe  
D = 10325476  
E = c3d2e1f0

**Basic Steps...**

**Pre-step4:** the 512-bit message is used to create a 5x512-bit chunk (see figure 5-9)
- n^th = (n-3 ⊕ n-8 ⊕ n-14 ⊕ n-16) \cdot 1 \quad (n \text{ starts from 16 to 79})
- 80 32-bit words

**Step 4:** the 80-step processing of 512-bit blocks – 4 rounds, 20 steps each.

**Each step** *t* (0 <= *t* <= 79):
- Input:
  - W_t – a 32-bit word from the message
  - K_t – a constant.
  - ABCDE: current MD.
- Output:
  - ABCDE: new MD.
Basic Steps...

- Only 4 per-round distinctive additive constants
  
  \[0 \leq t \leq 19: \quad K_t = 5A827999\]
  \[20 \leq t \leq 39: \quad K_t = 6ED9EBA1\]
  \[40 \leq t \leq 59: \quad K_t = 8F1BBCDC\]
  \[60 \leq t \leq 79: \quad K_t = CA62C1D6\]

Basic Logic Functions

- Only 3 different functions

<table>
<thead>
<tr>
<th>Round</th>
<th>Function (f(B,C,D))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 \leq t \leq 19</td>
<td>((B \land C) \lor \neg(B \land D))</td>
</tr>
<tr>
<td>20 \leq t \leq 39</td>
<td>(B \oplus C \oplus D)</td>
</tr>
<tr>
<td>40 \leq t \leq 59</td>
<td>((B \land C) \lor (B \land D) \lor (C \land D))</td>
</tr>
<tr>
<td>60 \leq t \leq 79</td>
<td>(B \oplus C \oplus D)</td>
</tr>
</tbody>
</table>

Twist With \(W_t\)’s

- Additional mixing used with input message 512-bit block
  \[W_0 \| W_1 \| \ldots \| W_{15} = m_0 | m_1 | m_2 | \ldots | m_{15}\]
  For \(15 < t < 80:\)
  \[W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}\]

- XOR is a very efficient operation, but with multilevel shifting, it should produce very extensive and random mixing!
SHA Versus MD5

- SHA is a stronger algorithm:
  - Brute-force birthday attacks requires on the order of $2^{80}$ operations vs. $2^{64}$ for MD5
  - SHA's 80 steps and 160 bits hash (vs. 128) requires a little more computation

- [http://www.cryptopp.com/benchmarks.html](http://www.cryptopp.com/benchmarks.html)