Lecture 5

Cryptographic Hash Functions

Read: Chapter 5 in KPS
Purpose

• CHF – one of the most important tools in modern cryptography and security

• In crypto, CHF instantiates a Random Oracle paradigm

• In security, used in a variety of authentication and integrity applications

• Not the same as “hashing” used in DB or CRCs in communications
Cryptographic HASH Functions

- **Purpose:** produce a fixed-size “fingerprint” or digest of arbitrarily long input data

- Why? To guarantee integrity

- Properties of a “good” cryptographic HASH function $H()$:
  1. Takes on input of any size
  2. Produces fixed-length output
  3. Easy to compute (efficient)
  4. Given any $h$, computationally infeasible to find any $x$ such that $H(x) = h$
  5. For a given $x$, computationally infeasible to find $y$ such that $H(y) = H(x)$ and $y \neq x$
  6. Computationally infeasible to find any $(x, y)$ such that $H(x) = H(y)$ and $x \neq y$
Same Properties Re-stated:

**Cryptographic properties of a “good” HASH function:**
- One-Way-ness (#4)
- Weak Collision-Resistance (#5)
- Strong Collision-Resistance (#6)

**Non-cryptographic properties of a “good” HASH function**
- Efficiency (#3)
- Fixed Output (#2)
- Arbitrary-Length Input (#1)
A hash function is typically based on an internal compression function $f()$ that works on fixed-size input blocks $(M_i)$.

Sort of like a Chained Block Cipher

- Produces a hash value for each fixed-size block based on (1) its content and (2) hash value for the previous block
- “Avalanche” effect: 1-bit change in input produces “catastrophic” and unpredictable changes in output
Simple Hash Functions

- Bitwise-XOR

<table>
<thead>
<tr>
<th>block 1</th>
<th>block 2</th>
<th>\cdots</th>
<th>block m</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_{11} )</td>
<td>( b_{21} )</td>
<td>\cdots</td>
<td>( b_{1m} )</td>
</tr>
<tr>
<td>( b_{12} )</td>
<td>( b_{22} )</td>
<td>\cdots</td>
<td>( b_{2m} )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>\cdots</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( b_{n1} )</td>
<td>( b_{n2} )</td>
<td>\cdots</td>
<td>( b_{nm} )</td>
</tr>
</tbody>
</table>

- Not secure, e.g., for English text (ASCII<128) the high-order bit is almost always zero
- Can be improved by rotating the hash code after each block is XOR-ed into it
- If message itself is not encrypted, it is easy to modify the message and append one block that would set the hash code as needed
- Another weak hash example: IP Header CRC
Another Example

- IPv4 header checksum
- One’s complement of the one’s complement sum of the IP header's 16-bit words
The Birthday Paradox

- Example hash function: \( y = H(x) \) where: \( x \) = person and \( H() \) is Bday()
- \( y \) ranges over set \( Y = [1...365] \), let \( n = \) size of \( Y \), i.e., number of distinct values in the range of \( H() \)
- How many people do we need to ‘hash’ to have a collision?
- Or: what is the probability of selecting at random \( k \) DISTINCT numbers from \( Y \)?
- probability of no collisions:
  - \( P_0 = 1 \times (1-1/n) \times (1-2/n) \times ... \times (1-(k-1)/n) = e^{(k-k)/2n} \)
- probability of at least one:
  - \( P_1 = 1 - P_0 \)
- Set \( P_1 \) to be at least 0.5 and solve for \( k \):
  - \( k = 1.17 \times \text{SQRT}(n) \)
  - \( k = 22.3 \) for \( n=365 \)

So, what’s the point?
The Birthday Paradox

\[ m = \log(n) = \text{size of } H() \]

\[ \sqrt{2^m} = 2^{m/2} \] trials must be computationally infeasible!
How Long Should a Hash be?

• Many input messages yield the same hash
  • e.g., 1024-bit message, 128-bit hash
  • On average, $2^{896}$ messages map into one hash
• With m-bit hash, it takes about $2^{m/2}$ trials to find a collision (with $\geq 0.5$ probability)
• When $m=64$, it takes $2^{32}$ trials to find a collision (doable in very little time)
• Today, need at least $m=160$, requiring about $2^{80}$ trials
# Hash Function Examples

<table>
<thead>
<tr>
<th></th>
<th>SHA-1 (weak)</th>
<th>MD5 (defunct)</th>
<th>RIPEMD-160 (unloved) 😊</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digest length</td>
<td>160 bits</td>
<td>128 bits</td>
<td>160 bits</td>
</tr>
<tr>
<td>Block size</td>
<td>512 bits</td>
<td>512 bits</td>
<td>512 bits</td>
</tr>
<tr>
<td># of steps</td>
<td>80 (4 rounds of 20)</td>
<td>64 (4 rounds of 16)</td>
<td>160 (5 paired rounds of 16)</td>
</tr>
<tr>
<td>Max msg size</td>
<td>$2^{64}$-1 bits</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

Other (stronger) variants of SHA are **SHA-256** and **SHA-512**
See: http://en.wikipedia.org/wiki/SHA_hash_functions
MD5

- Author: R. Rivest, 1992
- 128-bit hash
  - based on earlier, weaker MD4 (1990)
- Collision resistance (B-day attack resistance)
  - only 64-bit
- Output size not long enough today (due to various attacks)
MD5: Message Digest Version 5

Input Message

Output: 128-bit Digest
Overview of MD5

Figure 5-8. Overview of MD4, MD5, SHA-1
MD5 Padding

- Given original message M, add padding bits “100...” such that resulting length is 64 bits less than a multiple of 512 bits.
- Append *original length in bits* to the padded message
- Final message chopped into 512-bit blocks

*Figure 5-7. Padding for MD4, MD5, SHA-1*
MD5: Padding

Input Message

512 bit Block

Padding

Initial Value

MD5

Transformation Block by Block

Output: 128-bit Digest

Final Output
MD5 Blocks
MD5 Box

512-bit message chunks (16 words)

Initial 128-bit vector

128-bit result

F(x,y,z) = (x \land y) \lor (~x \land z)
G(x,y,z) = (x \land z) \lor (y \land \sim z)
H(x,y,z) = x \oplus y \oplus z
I(x,y,z) = y \oplus (x \land \sim z)

x\leftarrow y: x \text{ left rotate } y \text{ 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: 
  \[ m_0 | m_1 | m_2 \ldots | m_{15} \]
  - Digest MD$_0$ 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
Processing of Block $m_i$ - 4 Passes

Convention:
- $A - d_0$ ; $B - d_1$
- $C - d_2$ ; $D - d_3$
- $T_i$ : diff. constant

$$m_i$$

$ABCD=f_F(ABCD,m_i,T[1..16])$

$ABCD=f_G(ABCD,m_i,T[17..32])$

$ABCD=f_H(ABCD,m_i,T[33..48])$

$ABCD=f_I(ABCD,m_i,T[49..64])$

$MD_{i+1}$
Different Passes ...

• Different functions and constants

• Different set of $m_i$-s

• Different sets of shifts
Functions and Random Numbers

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

- SHA-0 was published by NIST in 1993

- Revised in 1995 as SHA-1
  - Input: Up to $2^{64}$ bits
  - Output: 160 bit digest
  - 80-bit collision resistance
  - Pad with at least 64 bits to resist padding attack
    - $1000 \ldots 0 \mid \mid \langle \text{message length} \rangle$

- Processes 512-bit block
  - Initiate 5x32bit MD registers
  - Apply compression function
    - 4 rounds of 20 steps each
    - each round uses different non-linear function
    - registers are shifted and switched

Figure 3.5 SHA-1 Processing of a Single 512-bit Block
Digest Generation with SHA-1
SHA-1 of a 512-Bit Block

Figure 3.5 SHA-1 Processing of a Single 512-bit Block
General Logic

• Input message must be $< 2^{64}$ bits
  • not a real limitation
• Message processed in 512-bit blocks sequentially
• Message digest (hash) 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 ...

- Step 4: the 80-step processing of 512-bit blocks: 4 rounds, 20 steps each
- Each step $t$ ($0 \leq t \leq 79$):
  - Input:
    - $W_t$ – 32-bit word from the message
    - $K_t$ – 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 Steps – Zooming In
## Basic Logic Functions

**Only 3 different functions**

<table>
<thead>
<tr>
<th>Round</th>
<th>Function $f_t(B, C, D)$</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
  - $W_0 | W_1 | ... | W_{15} = m_0 | m_1 | m_2 | ... | m_{15}$
  - 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 produces very extensive and random mixing!
SHA-1 Versus MD5

• SHA-1 is a stronger algorithm:
  • A birthday attack requires on the order of $2^{80}$ operations, in contrast to $2^{64}$ for MD5

• SHA-1 has 80 steps and yields a 160-bit hash (vs. 128) - involves more computation
Summary:
What are hash functions good for?
Message Authentication Using a Hash Function

Use symmetric encryption such as AES or 3-DES

- Generate $H(M)$ of same size as $E()$ block
- Use $E_K(H(M))$ as the MAC (instead of, say, DES MAC)
- Alice sends $E_K(H(M))$, $M$
- Bob receives $C,M'$ decrypts $C$ with $k$, hashes result
  \[ H(D_K(C)) =?= H(M') \]

Collision ➔ MAC forgery!
Using Hash for Authentication

Alice and Bob share a secret key $K_{AB}$

1. Alice $\rightarrow$ Bob: random challenge $r_A$
2. Bob $\rightarrow$ Alice: $H(K_{AB} || r_A)$, random challenge $r_B$
3. Alice $\rightarrow$ Bob: $H(K_{AB} || r_B)$

Only need to compare $H()$ results
Using Hash to Compute MAC: integrity

• Cannot just compute and append $H(m)$

• Need “Keyed Hash”:
  • Prefix:
    • $MAC: H(K_{AB} \mid m)$, almost works, but ...
    • Allows concatenation with arbitrary message:
      • $H(K_{AB} \mid m \mid m')$
  • Suffix:
    • $MAC: H(m \mid K_{AB})$, works better, but what if $m'$ is found such that $H(m) = H(m')$?

• HMAC:
  • $H(K_{AB} \mid H(K_{AB} \mid m))$
Hash Function MAC (HMAC)

- **Main Idea**: Use a MAC derived from any cryptographic hash function
  - hash functions do not use a key, therefore cannot be used directly as a MAC

- **Motivations for HMAC**:
  - Cryptographic hash functions execute faster in software than encryption algorithms such as DES
  - No need for the reverseability of encryption
  - No US government export restrictions (was important in the past)

- **Status**: designated as mandatory for IP security
  - Also used in Transport Layer Security (TLS), which will replace SSL, and in SET
HMAC Algorithm

- Compute $H1 = H()$ of the concatenation of $M$ and $K1$
- To prevent an “additional block” attack, compute again $H2 = H()$ of the concatenation of $H1$ and $K2$
- $K1$ and $K2$ each use half the bits of $K$
- Notation:
  - $K^+ = K$ padded with 0’s
  - $ipad = 00110110 \times b/8$
  - $opad = 01011100 \times b/8$
- Execution:
  - Same as $H(M)$, plus 2 blocks
Just for Fun…
Using a Hash to Encrypt

• (Almost) One-Time Pad: similar to OFB
  • compute bit streams using $H()$, $K$, and IV
    • $b_1 = H(K_{AB} \mid IV)$, ..., $b_i = H(K_{AB} \mid b_{i-1})$, ...
    • $c_1 = p_1 \oplus b_1$, ..., $c_i = p_i \oplus b_i$, ...

• Or, mix in the plaintext
  • similar to cipher feedback mode (CFB)
    • $b_1 = H(K_{AB} \mid IV)$, ..., $b_i = H(K_{AB} \mid c_{i-1})$, ...
    • $c_1 = p_1 \oplus b_1$, ..., $c_i = p_i \oplus b_i$, ...