Cryptographic Hash Functions

Guevara Noubir
http://www.ccs.neu.edu/home/noubir/Courses/CSG252/F04

Textbook: "Cryptography: Theory and Applications",
Douglas Stinson, Chapman & Hall/CRC Press, 2002

Reading: Chapter 4

Outline

- Applications of Hash Functions
- Definition of Hash Functions
- Security of Hash Functions
- Iterated Hash Functions (e.g., SHA)
- Message Authenticated Codes

Applications of Hash Functions

- Building Integrity Protection Services
- Files integrity
- Message Integrity

Definition

- Hash Family is a four-tuple (X, Y, K, H) s.t.
  - X is a set of possible messages (not necessarily finite)
  - Y is a finite set of possible message digests
  - K is a finite set of possible keys
  - For each $k \in K$, there exists a hash function $h_k \in H$ and $h_k : X \rightarrow Y$
  - $|X| = N$; $|Y| = M$;
  - $F^{X, Y}$ is the set of all functions from X to Y
  - $|F^{X, Y}| = M^N$
  - Any $F \subseteq F^{X, Y}$ is called an (N, M) family

Security of Hash Functions

- Preimage Problem:
  - Given a hash function $h: X \rightarrow Y$, and an element $y \in Y$
  - Find: $x \in X$ s.t. $h(x) = y$
- Second Preimage Problem:
  - Given a hash function $h: X \rightarrow Y$, and an element $x \in X$
  - Find: $x' \in X$ s.t. $x \neq x'$ and $h(x') = h(x)$
- Collision Problem:
  - Given a hash function $h: X \rightarrow Y$
  - Find: $x, x' \in X$ s.t. $x \neq x'$ and $h(x') = h(x)$

Random Oracle Model (RO)

- Definition:
  - Tries to capture the concept of "ideal" hash function
  - The only way to determine the value of the hash function at x is by evaluating h
  - Examples of function that do not satisfy the RO model:
    - $h(x, y) = ax + by \bmod n$
- Fundamental theorem:
  - Suppose that $h \in F^{X, Y}$ is chosen randomly, and let $X_0 \subseteq X$ s.t. $h(x)$ is known iff $x \in X_0$. Then for all $x \notin X_0$: $\Pr[h(x)=y] = 1/M$
- Las Vegas Randomized Algorithms:
  - Either returns a correct answer or returns "failure"

Hash Function Security

- FindPreImage(h, y, q)
  - Choose $X_0 \subseteq X$, $|X_0| = q$
  - For each $x \in X_0$ do
    - If $h(x) = y$ then return(x)
  - return(failure)
  - FindPreImage has average case success probability 1- (1-1/M)^q
- FindSecondPreimage
  - $Y \leftarrow h(x)$
  - Choose $X_0 \subseteq X\backslash\{x\}$; , $|X_0| = q$
  - For each $x \in X_0$ do
    - If $h(x) = y$ then return(x)
  - return(failure)
  - FindPreImage has average case success probability 1- (1-1/M)$^{q-1}$
- FindCollision
  - Choose $X_0 \subseteq X$, $|X_0| = q$
  - For each $x \in X_0$ do $y_x \leftarrow h(x)$
  - If $y_x = y_{x'}$ for $x \neq x'$ then return(x, x') else return(failure) $\varepsilon = 1 - (\frac{M-1}{M})(\frac{M-2}{M})...(\frac{M-q+1}{M})$
  - FindPreImage has average case success probability
  - $q \approx \sqrt{2M \ln \frac{1}{1-\varepsilon}}$

Comparison of Security Criteria

- CollisionToSecondPreImage(h)
  - **External** Oracle2ndPreImage
  - Choose $x \in X$ (random uniform)
  - If (Oracle2ndPreImage(h, x) = x') and ($x \neq x'$) and ($h(x) = h(x')$)
  - Then return(x, x')
  - Else return(failure)
- CollisionToPreImage(h)
  - **External** OraclePreImage
  - Choose $x \in X$ (random uniform); $y \leftarrow h(x)$;
  - If (OraclePreImage(h, y) = x') and ($x \neq x'$)
  - Then return(x, x')
  - Else return(failure)
- If $|X|>2|Y|$ and OraclePreImage is a (1, q)-algorithm for PreImage of h, then CollisionPreImage is a (1/2, q+1)

Iterated Hash Functions

- Goal:
  - Construct a hash function with infinite domain using finite domain hash function
  - $h: \bigcup_{l=m+t}^{\infty} \{0,1\}^l \rightarrow \{0,1\}^t$
- Assumptions:
  - Only consider bitstreams
  - *Compress*: $\{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$
- Algorithm Outline:
  - Input: bitstring x s.t. $|x| > m+t$
  - Preprocessing (usually achieved through padding):
    - Construct: $y = y_1 | y_2 | ... | y_r$
    - s.t. $|y| = 0 \pmod t$ and $|y_i| = t$
  - Processing:
    - $z_0 \leftarrow$ IV
    - $z_1 \leftarrow$ compress($z_0 | y_1$)
    - ...
    - $z_r \leftarrow$ compress($z_{r-1} | y_r$)
  - Optional Output Transformation:
    - Let g: $\{0, 1\}^m \rightarrow \{0, 1\}^l$
    - $h(x) = g(z_r)$

Merkle-Damgard

- Merkle-Damgard(x)
  - **External** compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ s.t. $t > 1$
  - $n \leftarrow |x|$
  - $k \leftarrow \lceil n/(t-1) \rceil$
  - $d \leftarrow n-k(t-1)$
  - for $i \leftarrow 1$ to $k-1$ do $y_i \leftarrow x_i$
  - $y_k \leftarrow x_k \mid 0^d$
  - $y_{k+1} \leftarrow$ binary representation of d
  - $z_1 \leftarrow 0^{m+1} \mid y_1$
  - $g_1 \leftarrow \text{compress}(z_1)$
  - for $i = 1$ to k
    - $z_{i+1} \leftarrow g_i \mid 1 \mid y_{i+1}$
    - $g_{i+1} \leftarrow \text{compress}(z_{i+1})$
  - return $(h(x) \leftarrow g_{k+1})$
- Theorem:
  - If compress is a collision resistant compression function where t>1 then *h* is a collision resistant hash function.

Merke-Damgard2(x)

- Let f s.t. f(0) = 0 and f(1) = 01
- Merkle-Damgard2(x)
  - **External** compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$
  - $n \leftarrow |x|$
  - $y \leftarrow 11 \mid f(x_1) \mid f(x_2) \mid \ldots \mid f(x_n)$
  - $g_1 \leftarrow \text{compress}(0^m \mid y_1)$
  - for $i \leftarrow 1$ to $k-1$ do $g_{i+1} \leftarrow \text{compress}(g_i \mid y_{i+1})$
  - return(g_k)
- Theorem:
  - If compress is a collision resistant compression function then *h* is a collision resistant hash function.

Secure Hash Algorithm-1

- **SHA-1-PAD**
  - Assumption: $L = |x| < 2^{64}-1$ (L is encoded on 64 bits)
  - $y \leftarrow x \mid 1 \mid 0^d \mid L$;
    - *d is s.t.* $|y|$ is a multiple of 512
- **SHA-1:**
  - h0 = 0x67452301; h1 = 0xEFCDAB89; h2 = 0x98BADCFE;
  - h3 = 0x10325476; h4 = 0xC3D2E1F0;
  - **(Process the message in successive 512-bit chunks:)**
  - **while** 512-bit chunk(s) remain(s)**:**
    - **break** the current chunk into sixteen 32-bit words w(i), 0 <= i <= 15
    - **(Extend the sixteen 32-bit words into eighty 32-bit words:)**
    - **for** i **from** 16 **to** 79: w(i) = (w(i-3) xor w(i-8) xor w(i-14) xor w(i-16)) leftrotate 1
    - a = h0; b = h1; c = h2; d = h3; e = h4;
    - **(Main loop:)**
      - **for** i from 0 **to** 79:
        - temp = (a leftrotate 5) + fi(b,c,d) + e + ki + w(i) *(note: all addition is mod 2^32)*
        - e = d; d = c; c = b leftrotate 30; b = a; a = temp;
      - h0 = h0 + a; h1 = h1 + b; h2 = h2 + c; h3 = h3 + d; h4 = h4 + e
  - **Return(** h0 | h1 | h2 | h3 | h4**) : 160 bits**
- (0 <= i <= 19): **fi**(b,c,d) = (b and c) or ((not b) and d), **ki** = 0x5A827999
- (20 <= i <= 39): **fi**(b,c,d) = (b xor c xor d), **ki** = 0x6ED9EBA1
- (40 <= i <= 59): **fi**(b,c,d) = (b and c) or (b and d) or (c and d), **ki** = 0x8F1BBCDC
- (60 <= i <= 79): **fi**(b,c,d) = (b xor c xor d), **ki** = 0xCA62C1D6

Message Authentication Codes

- Goal:
  - Construct keyed MACs to prevent fabrication and substitution
- Example of insecure MAC:
  - Using the key for IV
  -
- Nested-MAC
  - Goal: construct a secure "big MAC" using two keyed hash functions that satisfy some security constraints
  - Nested-MAC composes: (X, Y, K, G) with (Y, Z, L, H)
  - Constraints:
    - Y, and Z are finite sets s.t. |Y| > |Z|
    - (Y, Z, L, H) is a secure MAC given a fixed (unknown) key (secure "little MAC")
    - (X, Y, K, G) is collision resistant, given a fixed (unknown) key
  - Attacks:
    - Forger for the nested MAC (big MAC attack):
      - (K, L) are secret, attacker is allowed to query the Nested-MAC for $h_L(g_k(x))$ and tries to output (x', z) s.t. $h_L(g_k(x')) = z$
    - Forger for the little MAC:
      - L is secret, attacker is allowed to query the little-MAC for $h_L(y)$ and tries to output (y', z) s.t. $h_L(y') = z$
    - Collision finder for the hash family (unknown key):
      K is secret, attacker is allowed to query the a hash oracle for $g_k(x)$ and tries to output (x', x") s.t. $g_k(x') = g_k(x")$

- Theorem:
  - Suppose (X, Z, M, GoH) is a Nested-MAC.
  - Suppose there does not exist an $(\varepsilon_1, q+1)$-collision attack for a randomly chosen function $g_k \in G$, where the key is secret.
  - Suppose there does not exist an $(\varepsilon_2, q+1)$-forger for a randomly chosen function $h_L \in H$ where L is secret.
  - If an (ε, q)-forger exists for the nested-MAC, for a randomly chosen function $(goh)_{(K, L)} \in GoH$, then $\varepsilon \leq \varepsilon_1 + \varepsilon_2$
- Example of Nested MAC: HMAC
  - $HMAC_K(x)$ = SHA-1((K⊕*opad*) | SHA-1((K⊕*opad*)|x))
  - *ipad* = 3636…36; *opad* = 5C5C…5C
  - Assumption: SHA-1 restricted to one application is a secure MAC

CBC-MAC

- CBC-MAC: very popular technique
  - Use a block cipher in CBC mode
  - CBC-MAC is the result of the last encryption
  - Best known attack: birthday attack
    - $x_3, \ldots, x_n$ fixed bitstrings of length t (n>2)
    - $q \approx 1.172^{t/2}$;
    - $x_1^1, x_1^2, \ldots, x_1^q$: distinct t-bitstrings
    - $x_2^1, x_2^2, \ldots, x_2^q$: random t-bitstrings
    - $x^i = x_1^i \mid x_2^i \mid x_3 \mid \ldots \mid x_n$
    - $i \neq j \Rightarrow x^i \neq x^j$
    - $h_K(x^i) = h_K(x^j) \Rightarrow y_1^i \oplus x_2^i = y_1^j \oplus x_2^j$
    - $\Rightarrow$ v = $x_1^i \mid x_2^i \oplus x_2^j \mid x_3 \mid \ldots \mid x_n$ and v = $x_1^j \mid x_2^j \oplus x_2^j \mid x_3 \mid \ldots \mid x_n$ have the same CBC-MAC

- Unconditionally Secure MAC
  - Goal:
    - Construct MAC for which there does not exist a (ε, 0) and (ε, 1)-forgers
  - Assumption:
    - Users use a key for one MAC (similar to one time pad)
  - (ε, 0) –forgers $\Rightarrow$ fabrication
  - (ε, 1)-forgers $\Rightarrow$ substitution