Cryptographic Data Integrity

Week 7

Stallings: Ch 11, 12, 13
What do we cover

- Crypto Hashes
- Message Authentication Codes
Hash Functions: Main Idea

- Hash function $H$ is a lossy compression function
  - **Collision**: $H(x) = H(x')$ for some inputs $x \neq x'$
- $H(x)$ should look “random”
  - Every bit (almost) equally likely to be 0 or 1
- A cryptographic hash function must have certain properties

Some slides stolen from Vitaly Shmatikov
Property 1: One-Way

- Intuition: hash should be hard to invert
  - “Preimage resistance”
  - Given a random, it should be hard to find any $x$ such that $h(x)=y$
    - $y$ is an $n$-bit string randomly chosen from the output space of the hash function
    - $y=h(x')$ for some $x'$
Property 1: One-Way (cont’d)

- How hard?
  - Brute-force: try every possible x, see if h(x)=y
  - SHA-1 (a common hash function) has 160-bit output
    - Assuming $2^{34}$ trials per second, can do $2^{89}$ trials per year
    - Will take $2^{71}$ years to invert SHA-1 on a random image
Birthday Paradox

- T people
- Suppose each birthday is a random number taken from K days (K=365) – how many possibilities?
  - $K^T$ - samples with replacement
- How many possibilities that are all different?
  - $(K)_T = K(K-1)...(K-T+1)$ - samples without replacement
- Probability of no repetition?
  - $(K)_T/K^T \approx 1 - T(T-1)/2K$
- Probability of repetition?
  - $O(T^2)$
Property 2: Collision Resistance

- Should be hard to find \( x \neq x' \) such that \( h(x) = h(x') \)
- Birthday paradox
  - Let \( T \) be the number of values \( x, x', x'' \ldots \) we need to look at before finding the first pair \( x \neq x' \) s.t. \( h(x) = h(x') \)
  - Assuming \( h \) is random, what is the probability that we find a repetition after looking at \( T \) values? \( \mathcal{O}(T^2) \)
- Total number of values? \( \mathcal{O}(2^n) \)
  - \( n \) = number of bits in the output of hash function
- Conclusion: \( T \approx \mathcal{O}(2^{n/2}) \)
- Brute-force collision search is \( \mathcal{O}(2^{n/2}) \), not \( \mathcal{O}(2^n) \)
- For SHA-1, this means \( \mathcal{O}(2^{80}) \) vs. \( \mathcal{O}(2^{160}) \)
One-Way vs. Collision Resistance

- One-wayness does not imply collision resistance
  - Suppose g() is one-way
  - Define h(x) as g(x’) where x’ is x except the last bit
    - x = x₁..xₙ₋₁xₙ thus x’ = x₁..xₙ₋₁
  - h is one-way (cannot invert h without inverting g)
  - Collisions for h are easy to find: for any x, h(x₀)=h(x₁)
Collision resistance does not imply one-wayness

Suppose \( g() \) is collision-resistant
Define \( h(x) \) to be 0x if \( x \) is \((n-1)\)-bit long, else 1g(x)

Collisions for \( h \) are hard to find:
- If \( y \) starts with 0, then there are no collisions
- If \( y \) starts with 1, then must find collisions in \( g \)

\( h \) is not one way
Half of all \( y \)'s (those whose first bit is 0) are easy to invert (how?)
Thus random \( y \) is invertible with prob. 1/2
Weak Collision Resistance

- Given a randomly chosen $x$, hard to find $x'$ such that $h(x) = h(x')$
  - Attacker must find collision for a specific $x$...
  - Collision resistance: sufficient to find any collision
  - Brute-force attack requires $O(2^n)$ time
- Weak collision resistance does not imply collision resistance
Hashing vs. Encryption

- Hashing is one-way. There is no “un-hashing”!
  - A ciphertext can be decrypted with a decryption key… hashes have no equivalent of “decryption”
- Hash(x) looks “random”, but can be compared for equality with Hash(x’)
  - Hash the same input twice → same hash value
  - Encrypt the same input twice → different ciphertexts
Example: MD5

\[
\text{md5\_digest("The quick brown fox jumps over the lazy dog") = 9e107d9d372bb6826bd81d3542a419d6}
\]

\[
\text{md5\_digest("The quick brown fox jumps over the lazy cog") = 1055d3e698d289f2af8663725127bd4b}
\]
Overview of MD5

- Designed in 1991 by Ron Rivest
- Iterative design using compression function
History of MD5

- 2004: first collision attack
  - The only difference between colliding messages is 128 random-looking bytes
- 2007: chosen-prefix collisions
  - For any prefix, can find colliding messages that have this prefix and differ up to 716 random-looking bytes
- 2012: MD5 collisions used in cyberwarfare
  - Flame malware uses an MD5 prefix collision to fake a Microsoft digital code signature
Basic Structure of SHA-1

Split message into 512-bit blocks

Compression function
- Applied to each 512-bit block and current 160-bit buffer
- This is the heart of SHA-1

160-bit buffer (5 registers) initialized with magic values

Message length ($K \mod 2^{64}$)

Padding (1 to 512 bits)
SHA-1 Compression Function

Current message block

Current buffer (five 32-bit registers A,B,C,D,E)

Four rounds, 20 steps in each

Let’s look at each step in more detail...

Similar to a block cipher, with message itself used as the key for each round

Fifth round adds the original buffer to the result of 4 rounds

Buffer contains final hash value

Similar to a block cipher, with message itself used as the key for each round

Fifth round adds the original buffer to the result of 4 rounds

Buffer contains final hash value
One Step of SHA-1

- Logic function for steps:
  - $(B \land C) \lor \lnot (B \land D)$ for steps 0..19
  - $B \lor C \oplus D$ for steps 20..39
  - $(B \land C) \lor (B \land D) \lor (C \land D)$ for steps 40..59
  - $B \oplus C \oplus D$ for steps 60..79

- Current message block mixed in:
  - For steps 0..15, $W_{0..15} = $message block
  - For steps 16..79,
    
    \[ W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3} \]

- Multi-level shifting of message blocks:
  - 30 bitwise left-rotate
  - Special constant added (same value in each 20-step round, 4 different constants altogether)
How Strong is SHA-1

- Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- Brute-force inversion requires $2^{160}$ ops, birthday attack on collision resistance requires $2^{80}$ ops
- Some weaknesses discovered in 2005
  - Collisions can be found in $2^{63}$ ops
Applications of Hashes

- Message Authentication
- Digital Signatures
- Password files
- Intrusion detection and virus detection
Message Authentication

- Why?
  - Prove the integrity of a message

- Message M
  - Sender generates M
  - Receiver wants to ensure that message received is the same as M

- Sender and Receiver share a symmetric key K
Why?

1. "Hi, I'm Alice"

2. Intercept

3. "Hi, I'm Alice. I owe you $1,000,000"

Alice

Bob

Malory
Why – Network Attacks

- Message disclosure
- Traffic Analysis
- Masquerade: pretend to be Alice
- Content modification
- Sequence modification
- Timing modification
- Source repudiation
- Destination repudiation

Use Encryption!

Message Authentication!

Digital Signatures!
Example 1: Authentication (Sender)

\[ E(K, [M \ || \ H(M)]) \]
Example 1: How to Verify? (Receiver)

\[ E(K, [M || H(M)]) \]

\( M \) (L bits)

Decryption Algorithm

Hash value \( h_1 \)

Hash \( H \)

Hash \( h_2 \)

\( h_1 = h_2? \)
Example 2: Message Authentication

\[ M (L \text{ bits}) \rightarrow \text{Hash } H \rightarrow \text{Hash value} \rightarrow E(K, H(M)) \rightarrow M (L \text{ bits}) \]

Key K

Encryption Algorithm
Example 2: How to Verify?

$E(K, H(M))$

Key $K$

Hash $H$

Decryption Algorithm

Hash value $h_1$

$h_1 = h_2$?

$M$ (L bits)

Hash value $h_2$
Hashes for Digital Signatures

- Why?
  - Messages to sign can be long
  - Signature algorithms are slow
  - Hashes have constant output length and are fast

- So...
  - Hash first, then sign
Signature Generation

- \( M \) (\( L \) bits)
- Hash \( H \)
- Hash value \( h_1 \)
- Signature Algorithm
- \( S[\text{priv}K,H(M)] \)
- Private Key
Signature Verification

$S[\text{privK}, H(M)]$

Public Key

Verification Algorithm

Hash $H$

$M$ (L bits)

Hash value $h_2$

Verify?
Hashes for Password Files

- Passwords: access to any multi-user system
  - User ID – determines user privileges
  - Password

- How does the system store the passwords?
Record New Password

1. Generate password $pwd$
2. Generate random $salt$

Hash $H$:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Salt</th>
<th>$H(salt, pwd)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>salt</td>
<td>$H(salt, pwd)$</td>
</tr>
</tbody>
</table>
Password Verification

1. Write password $p$

Alice

User ID | Salt  | $H(salt, pwd)$
---|---|---
Alice | $salt$ | $H(salt, pwd)$

Password file

Hash $H$

$H(salt, p)$

Equal?
Intrusion and Virus Detection

- How do you ensure that file system has not been compromised?

- For each file F in the system
  - Compute H(F)
  - Store H(F) externally

- At boot-up
  - Compute H(F) for each file F
  - Compare against stored value