Hash Functions

Israel Koren

Content of this part

- Why we need hash functions?
- How do they work?
- Security properties
- Algorithms
  - Based on block ciphers
  - Other
- Example: The Secure Hash Algorithm SHA-1
**Motivation**

Problem: Naive signing of long messages generates a signature of same length.

- Computational overhead
- Message overhead
- Security issues

Solution:
Instead of signing the whole message, sign only a digest (=hash). Also secure, but much faster

Needed:
Hash Functions

---

**Digital Signature with a Hash Function**

\[ x_1 \mid x_2 \mid x_3 \ldots \mid x_n \]

\[ h(x) \]

\[ \text{Sig}_{k_{pr}}(z) \]

\[ y = \text{sig}_{k_{pr}}(z) \]

Notes:
- \( x_i \) has fixed length
- \( z, y \) have fixed length
- \( z, x \) do not have equal length in general
- \( h(x) \) does not require a key
- \( h(x) \) is public
Basic Protocol for Digital Signatures with a Hash Function

Alice

\( K_{pub} \)

Bob

\[ z = h(x) \]
\[ s = \text{sig}_{K_{pr}}(z) \]

\[ (x, s) \]

\[ z' = h(x) \]
\[ \text{ver}_{K_{pub}}(s, z') = \text{true/false} \]

Input-output behavior of hash functions

- Computationally efficient
- Fixed-length output
- Highly sensitive to input
Security properties of hash functions

1. Preimage resistance – one-wayness

- **Preimage resistance**: For a given output \( z \), it is impossible to find an input \( x \) such that \( h(x) = z \), i.e., \( h(x) \) is one-way.

- **Bob** sends \((e_K(x), \text{sig}_{K_{pr,B}}(z))\)
  Encrypts with AES and signs with RSA: \( s = \text{sig}_{K_{pr,B}}(z) = z^d \mod n \)

- **Oscar** uses Bob’s public key to calculate \( s^e = z \mod n \)
  If \( h(x) \) is not one-way then \( x = h^{-1}(z) \)

2. Second security properties of hash functions

- **Second preimage resistance**: Given \( x_1 \), and thus \( h(x_1) \), it is computationally infeasible to find an \( x_2 \) such that \( h(x_1) = h(x_2) \).

- Alice
  - \( k_{pub,B} \)

- Oscar
  - \( (x_1, s) \)

- Bob
  - \( h(x_1) = h(x_2) \)

  \( z = h(x_2) \)
  \( s = \text{sig}_{K_{pr,B}}(z) \)
  \( v_{K_{pub,B}}(s, z) = \text{true} \)

- There is always \( x_2 \) such that \( h(x_1) = h(x_2) \) but it should be difficult to find
- “weak” collision, requires exhaustive search
3rd security properties of hash functions

- **(Strong) Collision resistance**: It is computationally infeasible to find a pair \( x_1 \neq x_2 \) such that \( h(x_1) = h(x_2) \).
- \( x_1 = "\text{transfer }$10\text{ to Oscar's account}" \)
- \( x_2 = "\text{transfer }$10,000\text{ to Oscar's account}" \)

It turns out that collision resistance is more difficult to achieve

- How hard is it to find a collision with a probability of 0.5?
- Related Problem: How many people are needed such that two of them have the same birthday with a probability of 0.5?
- No! Not \( \frac{365}{2} = 183 \). 23 are enough. This is called the birthday paradox (search takes \( \approx \sqrt{2^n} \) steps)
- To deal with this paradox, hash functions need an output size of at least 160 bits

Hash function: Security

It turns out that collision resistance is more difficult to achieve

- How hard is it to find a collision with a probability of 0.5?
- Related Problem: How many people are needed such that two of them have the same birthday with a probability of 0.5?
- No! Not \( \frac{365}{2} = 183 \). 23 are enough. This is called the birthday paradox (search takes \( \approx \sqrt{2^n} \) steps)
- To deal with this paradox, hash functions need an output size of at least 160 bits

---

ECE597/697 Koren Part.11.9 Adapted from Paar & Pelzl, “Understanding Cryptography,” and other sources

ECE597/697 Koren Part.11.10 Adapted from Paar & Pelzl, “Understanding Cryptography,” and other sources
Hash Algorithms

- MD4 and MD5: families of Hash functions
- SHA-1: output - 160 Bit; input - 512 bit chunks of message $x$; operations - bitwise AND, OR, XOR, complement and cyclic shifts.
- RIPE-MD 160: output - 160 Bit; input - 512 bit chunks of message $x$; operations - like in SHA-1, but two in parallel and combinations of them after each round.

Hash based on a block cipher

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

- $H_0$ - fixed known value
- Example of block cipher: AES ($b=m$)

$H_i = H_{i-1} \oplus x_i \oplus e_g(H_{i-1})(x_i)$
Block cipher based Hash with $m=2b$

- Twice the block size
- Example – Hiroshi Hash
- AES with 256-bit key
- $c$ – non-zero constant

SHA-1

- Part of the MD-4 family.
- Based on a Merkle-Dåmgard construction.
- 160-bit output from a message of maximum length $2^{64}$ bit.
- Widely used (even though some weaknesses are known)
SHA-1 High Level Diagram

- Compression function consists of 80 rounds which are divided into four stages of 20 rounds each

\[ x = x_1 x_2 \ldots x_n \]

- \( H_0 \) - pre-defined constant

SHA-1: Padding

- Message \( x \) has to be padded to fit a size of a multiple of 512 bit.
- \( k = 512 - 64 - 1 - \ell = 448 - (\ell + 1) \mod 512. \)
- 64 bits for the binary representation of \( \ell \)
SHA-1: Hash Computation

- Each message block $x_i$ is processed in four stages with 20 rounds each (512 bits producing $160 = 32 \times 5$ bits)
- SHA-1 uses:
  - A message schedule which computes a 32-bit words $W_0, W_1, \ldots, W_{79}$ for each of the 80 rounds
  - Five working registers of size of 32 bits $A, B, C, D, E$
  - A hash value $H_i$ consisting of five 32-bit words $H_i^{(0)}, H_i^{(1)}, H_i^{(2)} , H_i^{(3)}, H_i^{(4)}$
  - In the beginning, the hash value holds the initial value $H_0$, which is replaced by a new hash value after the processing of each single message block.
  - The final hash value $H_n$ is equal to the output $h(x)$ of SHA-1.

\[
W_j = \begin{cases} 
 x_j^{(i)} & 0 \leq j \leq 15 \\
 (W_{j-16} \oplus W_{j-14} \oplus W_{j-13} \oplus W_{j-12} \oplus W_{j-11} \oplus W_{j-9} \oplus W_{j-8})_{16} & 16 \leq j \leq 79
\end{cases}
\]
SHA-1: Internals of a Round

<table>
<thead>
<tr>
<th>Stage t</th>
<th>Round j</th>
<th>Constant K,</th>
<th>Function f,</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 ... 19</td>
<td>K=5A827999</td>
<td>f(B,C,D)=(B ∧ C) ∨ (~B ∧ D)</td>
</tr>
<tr>
<td>2</td>
<td>20 ... 39</td>
<td>K=6ED9EBA1</td>
<td>f(B,C,D)=B ⊕ C ⊕ D</td>
</tr>
<tr>
<td>3</td>
<td>40 ... 59</td>
<td>K=8F1BBDCA</td>
<td>f(B,C,D)=(B ⊕ C) ∨ (B ⊕ D) ∨ (C ⊕ D)</td>
</tr>
<tr>
<td>4</td>
<td>60 ... 79</td>
<td>K=CA62C1D6</td>
<td>f(B,C,D)=B ⊕ C ⊕ D</td>
</tr>
</tbody>
</table>

Lessons Learned

- Hash functions are keyless. The two most important applications are: digital signatures and in message authentication codes such as HMAC.
- The three security requirements for hash functions are one-wayness, second preimage resistance and collision resistance.
- Hash functions should have at least 160-bit output length in order to withstand collision attacks; 256 bit or more is desirable for long-term security.
- MD5, which was widely used, is insecure. Some security weaknesses have been found in SHA-1, and it is being phased out. The SHA-2 algorithms appear to be secure.
- The SHA-3 competition will result in new standardized hash functions.