Content of this part

♦ Why do we need hash functions?
♦ How do they work?
♦ Security properties
♦ Algorithms
  ♦ Based on block ciphers
  ♦ Other
♦ Example: The Secure Hash Algorithm SHA-1
♦ SHA-3
**Motivation**

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

- **3 Problems**
  - 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**

\[ x_i = h(x_i || z_{i-1}) \]

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

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

\[ K_{pub} \]

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

Bob

\[ (x, s) \]

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

Input-output behavior of hash functions

<table>
<thead>
<tr>
<th>Alice was beginning to get very tired of sitting by her sister on the bank, and having nothing to do.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ h ]</td>
</tr>
<tr>
<td>[ DFDC349A ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I am not a crook.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ h ]</td>
</tr>
<tr>
<td>[ FB93E283 ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I am not a cook.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ h ]</td>
</tr>
<tr>
<td>[ A3F4439B ]</td>
</tr>
</tbody>
</table>

- 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 \( s^e \equiv z \mod n \)
  If \( h(x) \) is not one-way then \( x = h^{-1}(z) \)

2nd 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
  - \( k_{pub,B} \)
  - \( (x_1, s) \)

- Oscar
  - \( z = h(x_1) \)
  - \( s = \text{sig}_{k_{pr,B}}(z) \)
  - \( (x_1, s) \)

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

- 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 $10 to Oscar’s account”}$
  - $x_2 = \text{“transfer $10,000 to Oscar’s account”}$

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 $365/2 = 183$. 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 $365/2 = 183$. 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
• 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.

\[ H_i = e_{g(H_{i-1})}(x_i) \oplus x_i \]

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

![SHA-1 Diagram]

- $H_0$ - 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=32x5 bits)
- SHA-1 uses:
  - A message schedule which computes 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.

\[ A = H_0^{(0)} = 67452301 \]
\[ B = H_0^{(1)} = EFCDAB89 \]
\[ C = H_0^{(2)} = 98BADCFE \]
\[ D = H_0^{(3)} = 10325475 \]
\[ E = H_0^{(4)} = C3D2E1F0 \]
SHA-1: Internals of a Round

\[ A, B, C, D, E = (E + f_i(B, C, D) + (A_{\ll 5} + W_j + K_i), A, (B_{\ll 30}, C, D) \]

<table>
<thead>
<tr>
<th>Stage t</th>
<th>Round j</th>
<th>Constant K_t</th>
<th>Function f_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 ... 19</td>
<td>K=5A827999</td>
<td>f(B, C, D)=(B \land C) \lor (\neg B \land D)</td>
</tr>
<tr>
<td>2</td>
<td>20 ... 39</td>
<td>K=6ED9EBA1</td>
<td>f(B, C, D)=B \oplus C \oplus D</td>
</tr>
<tr>
<td>3</td>
<td>40 ... 59</td>
<td>K=8F1BBCDC</td>
<td>f(B, C, D)=(B \oplus C) \lor (B \oplus D) \lor (C \oplus D)</td>
</tr>
<tr>
<td>4</td>
<td>60 ... 79</td>
<td>K=CA62C1D6</td>
<td>f(B, C, D)=B \oplus C \oplus D</td>
</tr>
</tbody>
</table>

Predicted Demise of SHA-1

♦ In 2012 it was predicted that $2^{61}$ attempts will be needed to find a collision in SHA-1
♦ It was then predicted (based on Moore’s law) that in 2017-2018 such an attack will be possible to mount at a price of about $200K using GPUs
♦ SHA-2 (a greatly modified SHA-1) was released in 2001 and has several versions: SHA-224, SHA-256, SHA-384 and SHA-512
♦ In 2015 several attacks have been shown to be feasible on reduced complexity SHA-2 versions
SHA-3: History

- Competition started in November 2007 (by NIST)
- Requirements: Output hash values of 224, 256, 384, 512 bits (> 160)
- Similar process as the AES competition
- Maximum message length at least $2^{64} - 1$ bits
- Implementable in a wide range of hardware and software platforms
- Deadline for first round: October 2008
- 14 2nd round candidates announced: July 2009
- 5 finalists: Dec. 2010
- In October 2012: winner announced: Keccak by G. Bertoni, J. Daemen, M. Peeters, G. Van Assche
- August 2015: official publication by NIST of SHA-3

SHA-3 (Keccak – 2012)

- Based on a SPONGE function construction
- Absorbing phase: read in the padded input bit string
- Squeezing phase: output as many hash bits as required.
Sponge transformation function operates on a state $S$ of $b$ bits

- The state $S$ splits into: $S = R || C$
  - $R$ is of bit length $r$ ($r$ is the bit rate)
  - $C$ is of bit length $c$ ($c$ is the capacity)

Input $M$ is divided in blocks $M_i$ of $r$ bits and padded to a multiple of $r$: $M = M_1 || M_2 || \cdots || M_t$ (padding 10/1)

A sponge round updates the state $S \leftarrow f(R \oplus M_i || C)$

- $S$ initialized to 0
- Capacity part $C$ is not directly linked to the input during the absorbing phase or the output during the squeezing phase
- Capacity $c$ is linked to the resistance level against a collision attack - can stand an attack up to complexity $2^c/2$
- $r$ affects run time efficiency, $c$ controls security

---

SHA-3 (2)

- State is a 3-dimensional array: $S[x][y][z]$ has $b = 5 \times 5 \times 2^\ell$ bits $0 \leq \ell \leq 6$
  - $2^\ell$ word length: 32-bit processor: $\ell = 5$ yields a $25 \cdot 32 = 800$ bit state
  - 64-bit processor: $\ell = 6 \Rightarrow 25 \cdot 64 = 1600$ bits
- SHA-3 is Keccak-$f[1600]$
- Number of rounds: $12 + 2\ell$
- round comprises 5 steps:
  $S \leftarrow \iota \circ \chi \circ \pi \circ \rho \circ \theta(S)$
  - $\theta$: replace a bit by XOR of this bit and the parity of two neighboring columns - provides minor diffusion
  - $\rho$: rotate the bits in a lane
  - $\pi$: permute lanes (permutation in the $(x, y)$-plane)
SHA-3 (4)

- \( \chi \): replace a bit by combining it with 2 subsequent bits in a row
- Flip bit if neighbors exhibit 01 pattern - non-linear step
- \( \iota \): add a round constant to lane (break symmetry)
- SHA-3: \( r = 1088 \) and \( c = 512 \)
  - permutation width: 1600
  - security strength 256

SHA-3: security

<table>
<thead>
<tr>
<th>Function</th>
<th>Output Size</th>
<th>Security Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Collision</td>
<td>Preimage</td>
</tr>
<tr>
<td>SHA-1</td>
<td>&lt; 80</td>
<td>160</td>
</tr>
<tr>
<td>SHA-224</td>
<td>112</td>
<td>224</td>
</tr>
<tr>
<td>SHA-512/224</td>
<td>112</td>
<td>224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>SHA-512/256</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>192</td>
<td>384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>SHA3-224</td>
<td>112</td>
<td>224</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>SHA3-384</td>
<td>192</td>
<td>384</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>256</td>
<td>512</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 3 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.

• Some security weaknesses have been found in SHA-1, and it is being phased out. The SHA-2 algorithms appear to be more secure but also start to be questionable.

• The SHA-3 competition resulted in new standardized hash functions.