Hash Functions (and Message Authentication Codes)

- Guarantee integrity of information after the application of the function.
- A cryptographic hash function is keyless, a MAC has a key.
- A cryptographic hash function is usually used as a component of another scheme.

Hash Functions

- A cryptographic hash function h is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output, the *hash value*.
- A cryptographic hash function should be one-way: given any string y from the range of h, it should be computationally infeasible to find any value x in the domain of h such that

$$h(x) = y.$$

- Given a hash function with outputs of n bits, we would like a function for which finding preimages requires $O(2^n)$ time.

Hash Functions – Collision Resistant

- In practice we need something more than the one-way property.
- A hash function is called *collision resistant* if it is infeasible to find two distinct values x and x' such that

$$h(x) = h(x').$$

Hash Functions – Collision Resistant (Birthday Paradox)

- *Birthday paradox:* To find a collision of a hash function f, we can keep computing

$$f(x_1), f(x_2), f(x_3), \ldots$$

until we get a collision. Output size n bits, then we expect to find a collision after $O(2^{n/2})$ tries. (more later)

Hash Functions – Second Preimage Resistant

- Second preimage resistant: given x it should be hard to find an $x' \neq x$ with $h(x') = h(x)$.
- a cryptographic hash function with n-bit outputs should require $O(2^n)$ operations before one can find a second preimage.

Hash Functions – Requirements

- **Preimage Resistant:** It should be hard to find a message with a given hash value.
- **Second Preimage Resistant:** Given one message it should be hard to find another message with the same hash value.
- **Collision Resistant:** It should be hard to find two messages with the same hash value.

Lemma

Assuming a function is preimage resistant for almost every element of the range of h is a weaker assumption than assuming it either collision resistant or second preimage resistant.

Lemma

Assuming a function is second preimage resistant is a weaker assumption than assuming it is collision resistant.

Hash Functions – Finding Collisions (Birthday Paradox)

- Upper bound on complexity of finding collision.
- Probability of collision when we observe M blocks with n bits each.
- One block can take $N = 2^n$ values.
- Probability that blocks are distinct is

$$\left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\cdots\left(1 - \frac{M-1}{N}\right) \approx$$

$$\prod_{i=1}^{M-1} e^{-\frac{i}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{M-1} i} =$$

$$e^{-\frac{M(M-1)}{2N}}$$

since $e^x \approx 1 - x$ when x is small.

Hash Functions – Finding Collisions (Birthday Paradox)

- Probability for at least one collision

$$\begin{aligned} &\varepsilon \approx 1 - e^{-\frac{M(M-1)}{2N}} \Longrightarrow \\ &-\frac{M(M-1)}{2N} \approx \ln(1-\varepsilon) \Longrightarrow \\ &M^2 - M \approx 2N \ln \frac{1}{1-\varepsilon}. \end{aligned}$$

- Since M is large we can write

$$M \approx \sqrt{2N \ln \frac{1}{1-\varepsilon}}.$$

- Examples:

$$\begin{aligned} \varepsilon = 0.5 \quad &\Longrightarrow \quad M \approx 1.18\sqrt{N} \\ \varepsilon = 0.95 \quad &\Longrightarrow \quad M \approx 2.45\sqrt{N} \end{aligned}$$

Designing Hash Functions

- Designing functions of infinite domain is hard,
- one can build a so called *compression function*, which maps bits strings of length s into bit strings of length n, for $s > n$, and then chain this in some way to produce a function on an infinite domain.
- The most famous chaining method: *the Merkle-Damgård construction*.

Merkle-Damgård construction

- f is a compression function from s bits to n bits, $s > n$, believed to be collision resistant.
- use f to construct h which takes arbitrary length inputs.
- f collision resistant $\Rightarrow$ h collision resistant.

1. $l = s - n$. Pad m with zeros so it is a multiple of l bits, write $m = m_1 m_2 \cdots m_t$. Set C_0 to some fixed initial value.
2. **for** $i = 1$ **to** t **do** $C_i = f(C_{i-1} || m_i)$
3. Set $h(m) = C_t$.

Merkle-Damgård construction

- **Length strengthening:** input message is preprocessed by first padding with zero bits to obtain a message which has length a multiple of l bits. Then a final block of l bits is added which encodes the original length of the unpadded message in bits. The construction is limited to hashing messages with length less than 2^l bits.
- Theory: If f is collision resistant then so is h.

Constructions: The MD4 Family

Most widely deployed: MD5, RIPEMD-160 and SHA-1.

- MD4: 3 rounds of 16 steps and an output bitlength of 128 bits.
- MD5: 4 rounds of 16 steps and an output bitlength of 128 bits.
- SHA-1: 4 rounds of 20 steps and an output bitlength of 160 bits.
- RIPEMD-160: 5 rounds of 16 steps and an output bitlength of 160 bits.
- SHA-256: 64 rounds of single steps and an output bitlength of 256 bits.
- SHA-384: identical to SHA-512 except the output is truncated to 384 bits.
- SHA-512: 80 rounds of single steps and an output bitlength of 512 bits.

In recent years a number of weaknesses have been found in almost all of the early hash functions in the MD4 family, for example MD4, MD5 and SHA-1.

SHA-1

- the internal state of the algorithm is a set of five 32-bit values

$$(H_1, H_2, H_3, H_4, H_5).$$

- define four round constants y_1, y_2, y_3, y_4.
- The length strengthening method used:
  - first append a one bit to the message (to signal its end),
  - pad with zeros to a multiple of the block length (512 bits),
  - as a separate final block, add message length (in bits).

SHA-1

The data stream is loaded 16 words at a time into X_j for $0 \leq j < 16$.

Algorithm 10.4: SHA-1 Overview

$(A, B, C, D, E) = (H_1, H_2, H_3, H_4, H_5)$
/* Expansion */
for $j = 16$ **to** 79 **do**
  $X_j = ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \lll 1)$
end
Execute Round 1
Execute Round 2
Execute Round 3
Execute Round 4
$(H_1, H_2, H_3, H_4, H_5) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$

The output is the concatenation of the final value of H_1, H_2, H_3, H_4, H_5.

Algorithm 10.5: Description of the SHA-1 round functions

Round 1
for $j = 0$ **to** 19 **do**
$t = (A \lll 5) + f(B, C, D) + E + X_j + y_1$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 2
for $j = 20$ **to** 39 **do**
$t = (A \lll 5) + h(B, C, D) + E + X_j + y_2$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 3
for $j = 40$ **to** 59 **do**
$t = (A \lll 5) + g(B, C, D) + E + X_j + y_3$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 4
for $j = 60$ **to** 79 **do**
$t = (A \lll 5) + h(B, C, D) + E + X_j + y_4$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end

SHA-1

Three bit-wise functions of three 32-bit variables:

$$f(u, v, w) = (u \wedge v) \vee ((\neg u) \wedge w),$$

$$g(u, v, w) = (u \wedge v) \vee (u \wedge w) \vee (v \wedge w),$$

$$h(u, v, w) = u \oplus v \oplus w.$$

Three bit-wise functions of three 32-bit variables:

$$f(u,v,w) = (u \wedge v) \vee ((\neg u) \wedge w),$$

$$g(u,v,w) = \underbrace{(u \wedge v) \vee (u \wedge w)}_{=u \wedge (v \vee w)} \vee (v \wedge w),$$

$$h(u,v,w) = u \oplus v \oplus w.$$

Security status

- In practice, MD5 and SHA-1 are by far the most common.
- Both are Merkle-Damgård constructions.
- Both are broken.
    - MD5 in practice.
    - SHA-1 in theory.

Security status

- In practice, MD5 and SHA-1 are by far the most common.
- Both are Merkle-Damgård constructions.
- Both are broken.
    - MD5 in practice.
    - SHA-1 in theory.
- There is also a SHA-2 family of hash functions.
    - Still ok.

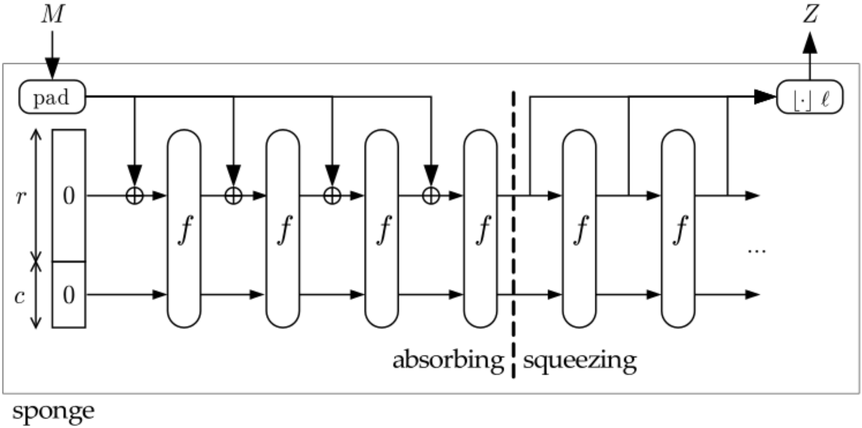
Security status

- In practice, MD5 and SHA-1 are by far the most common.
- Both are Merkle-Damgård constructions.
- Both are broken.
  - MD5 in practice.
  - SHA-1 in theory.
- There is also a SHA-2 family of hash functions.
  - Still ok.
- Newest family: SHA-3
  - Output of NIST competition 2012.

The SHA-3 standard

- On October 2, 2012, **Keccak** was selected as the winner.
- Keccak is a family of cryptographic hash functions designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche,
- SHA-3 uses the *sponge construction*, in which message blocks are XORed into the initial bits of the state

The SHA-3 standard



Usage of hash functions

- **Commit to message** by disclosing hash of message, later showing the message
  - If collision resistant, you cannot cheat (change message).
  - Consider playing rock, paper, scissors remotely with a hash function.
  - Or rock-paper-scissors-lizard-Spock.
- **Verify integrity** of downloaded files.
- **Digital signatures**.
- **SSL/TLS** for integrity protection.
- **Storing passwords** in operating systems and web servers.

Message Authentication Codes (MACs)

- Keyed hash function.
- **Authenticate origin of messages**
  - Symmetric key, shared between sender and receiver.
  - Both sender and receiver can create and verify MAC.
- **Integrity protection of messages**
  - Message changes in transit are detected.
  - An ordinary (key-less) hash function does not provide this. (why?)

Message Authentication Codes (MACs)

- Keyed hash function.
- **Authenticate origin of messages**
  - Symmetric key, shared between sender and receiver.
  - Both sender and receiver can create and verify MAC.
- **Integrity protection of messages**
  - Message changes in transit are detected.
  - An ordinary (key-less) hash function does not provide this. (why?)
- Two known designs:
  - HMAC (based on hash function)
  - CBC-MAC (based on block cipher in CBC-mode)

Message Authentication Codes (MACs)

- Keyed hash function.
- **Authenticate origin of messages**
  - Symmetric key, shared between sender and receiver.
  - Both sender and receiver can create and verify MAC.
- **Integrity protection of messages**
  - Message changes in transit are detected.
  - An ordinary (key-less) hash function does not provide this. (why?)
- Two known designs:
  - HMAC (based on hash function)
  - CBC-MAC (based on block cipher in CBC-mode)
- Are these good constructions?
  - $MAC_k(m) = h(k \| m)$.
  - $MAC_k(m) = h(m \| k)$.

Message Authentication Codes (MACs)

Is $MAC_k(m) = h(k \| m)$ a good construction?

- No!
- Assume we know

$$c = MAC_k(m_1) = h(k \| m_1).$$

- Then we can find MAC of message

$$MAC_k(m_1 \| pad_{m_1} \| m_2) = h(k \| m_1 \| pad_{m_1} \| m_2) = f(c, m_2)$$

without knowing the key.

Message Authentication Codes (MACs)

Is $MAC_k(m) = h(m\|k)$ a good construction?

- No!
- Find a collision in the hash function, such that

$$h(m_1) = h(m_2).$$

- Then

$$MAC_k(m_1) = MAC_k(m_2).$$

If h if collision resistant, then so is MAC_k.
But MACs can be built *without* requiring collision resistance in the underlying hash function.
HMAC has this property.

HMAC

HMAC is a MAC based on a hash function:

$$HMAC_k(m) = h\left((k \oplus opad) \| h\left((k \oplus ipad) \| m\right)\right).$$

- opad = 0x5c5c5c5c...
- ipad = 0x36363636...
- Proposed in 1996.
- Used with MD5 or SHA-1 in SSL/TLS.
- Immune to previous attacks.

Hash Functions from Block Ciphers.

- Pad the message to be hashed and divide it into blocks

$$x_0, x_1, \ldots, x_t,$$

- $H_0 = IV$, and iterate

$$H_i = f(x_i, H_{i-1}).$$

- For example, a **Davies-Meyer hash**

$$f(x_i, H_{i-1}) = E_{x_i}(H_{i-1}) \oplus H_{i-1}.$$