- 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.
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

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

- *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.
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.
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.
Designing Hash Functions

- Designing functions of infinite domain is hard,
- one builds a so called *compression function*, which maps bits strings of length $s$ into bit strings of length $n$, for $s > n$, and then chains 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.

1. $l = s - n$. Pad $m$ with zeros so it is a multiple of $l$ bits, write $m = m_1m_2 \cdots m_t$. Set $H$ to some fixed value.
2. for $i = 1$ to $t$ do $H = f(H|m_i)$
3. Set $h(m) = H$. 

Novembre 24, 2011 8 / 15
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.
• 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 is to first append a one bit to
the message, to signal its end and then to pad with zeros to a multiple
of the block length = 512 bits. Finally the number of bits of the
message is added as a separate final block.
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

$X_j = ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \ll 1)$

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
\[
\begin{align*}
    t &= (A \ll 5) + f(B, C, D) + E + X_j + y_1 \\
    (A, B, C, D, E) &= (t, A, B \ll 30, C, D)
\end{align*}
\]
end

Round 2
for \( j = 20 \) to 39 do
\[
\begin{align*}
    t &= (A \ll 5) + h(B, C, D) + E + X_j + y_2 \\
    (A, B, C, D, E) &= (t, A, B \ll 30, C, D)
\end{align*}
\]
end

Round 3
for \( j = 40 \) to 59 do
\[
\begin{align*}
    t &= (A \ll 5) + g(B, C, D) + E + X_j + y_3 \\
    (A, B, C, D, E) &= (t, A, B \ll 30, C, D)
\end{align*}
\]
end

Round 4
for \( j = 60 \) to 79 do
\[
\begin{align*}
    t &= (A \ll 5) + h(B, C, D) + E + X_j + y_4 \\
    (A, B, C, D, E) &= (t, A, B \ll 30, C, D)
\end{align*}
\]
end
Three bit-wise functions of three 32-bit variables:

\[ f(u, v, w) = (u \land v) \lor ((\neg u) \land w), \]

\[ g(u, v, w) = (u \land v) \lor (u \land w) \lor (v \land w), \]

\[ h(u, v, w) = u \oplus v \oplus w. \]
Pad the message to be hashed and divide it into blocks
\[ x_0, x_1, \ldots, x_t, \]

\[ H_0 = IV, \text{ and iterate} \]
\[ H_i = f(x_i, H_{i-1}). \]

For example, a **Davies-Meyer hash**
\[ f(x_i, H_{i-1}) = E_{x_i}(H_{i-1}) \oplus H_{i-1}. \]