Iterated Hash Functions

- Generic Construction Method
- The Merkle-Damgard Construction
- The Secure Hash Algorithm (SHA)
Generic Construction Method

- Suppose that we have already has a compression function, say, `compress` that maps from \( \{0,1\}^{m+t} \rightarrow \{0,1\}^m \).
- \( m = \) message digest size, \( t \geq 1 \).
- As usual,
  - \( |x| = \) length of a bitstring \( x \)
  - \( x \ || \ y = \) bitstring \( x \), \( y \) concatenation.
x = input bit string. assume |x| > m+t.

y = constructed from x with some padding function, say pad(x), so that

- y = x || pad(x) can be divided evenly in blocks of size t:
  - y = y₁ || y₂ || ... yᵣ.
  - |yᵢ| = length of yᵢ = t, for 1 ≤ i ≤ t.
Generic Construction
Processing Step

- **IV** = public initial value of length m.
- Use **compress** function to iteratively process the r blocks of \( y_i \)'s as in:
  - \( z_0 \leftarrow IV \)
  - \( z_1 \leftarrow compress(z_0 \ || \ y_1) \)
  - \( z_2 \leftarrow compress(z_1 \ || \ y_2) \)
  - ...
  - \( z_r \leftarrow compress(z_{r-1} \ || \ y_r) \)
The final hash function is defined as $h(x) = g(z_r)$, where
- $g: \{0,1\}^m \rightarrow \{0,1\}^l$ is a public function, it is optional.
- if no transformation required, $h(x) = z_r$. 

See Fig 4.1 (page 131) for illustration.

Merkle-Damgard Construction Method is an example of this type of iterative hash function with a formal security proof.
FIGURE 4.1
The processing step in an iterated hash function
Transforming $x \rightarrow y$

- Initial mapping of $x$ to $y$ must be 1-1.
- Why? If we can find $x \neq x'$ that $y = y'$, then $h(x) = h(x')$ as defined (why?).
- $h(x)$ is no longer a collision-resistant hash function even if we assume the compress function used is collision-resistant.
- $y = x \ || \ pad(x)$ is clearly 1-1. (why?)
Merkle-Damgard Construction Method

- Suppose that \( \text{compress}: \{0,1\}^{m+t} \rightarrow \{0,1\}^m \) is a collision-resistant function.
- Goal: Using this function, \( \text{compress} \), to construct a “provable” collision-resistant hash function, \( h(x) \), that maps from unbounded space, \( X \), to \( \{0,1\}^m \).
- We will consider (a) \( t \geq 2 \), then (b) \( t \geq 1 \).
Basic Step of Algorithm 4.6. Merkle-Damgard(x), \( t \geq 2 \).

- Let \( n = |x| \), divide \( x \) into blocks of size \((t-1)\), except (possibly) the last block. How many blocks?
  - \( k = \lceil \frac{n}{(t-1)} \rceil = \text{ceil}(n/(t-1)) \).

- \( x = x_1 \ || \ x_2 \ || \ ... \ || \ x_k \), pad the last block \( x_k \) with \( d \) copies of 0's. How many?
  - \( d = (t-1) - |x_k| = k(t-1) - n \).

- Define \( y_k \leftarrow x_k \ || \ 0^d \)
  - \( y_i \leftarrow x_i \), for \( i = 1,2,..,k-1 \).
  - \( y_{k+1} \leftarrow (t-1) \) bits binary representation of \( d \).

- \( y = y_1 \ || \ y_2 \ || \ ... \ || \ y_k \ || \ y_{k+1} \), each block of \( y_i \) is now of size \((t-1)\).
Simple illustration

- \( n = |x| = 123456 \)
- \( t = 100, \) block of size \((t-1)=99\)
- \( k = \text{ceil}(123456/99) = \text{ceil}(1247.03) = 1248 \)
- \( d = k(t-1)-n = 96 = 1100000_2 \)
- \( x' \leftarrow x_1 \parallel x_2 \parallel ... \parallel (x_{1248} \parallel 0^{96}) \)
- \( y \leftarrow x' \parallel (0^{92} \parallel 1100000) \)
- \( y = y_1 \parallel y_2 \parallel ... \parallel y_{1249}. \)
Algorithm 4.6. Merkle-Damgard(x), t ≥ 2.

- Suppose that \( \text{compress}: \{0,1\}^{m+t} \rightarrow \{0,1\}^m \) is a collision-resistant function.
- \( k = \lceil n/(t-1) \rceil, \ n = |x| \)
- Construct \( y = y_1 \parallel y_2 \parallel \ldots \parallel y_k \parallel y_{k+1} \)
- \( g_1 \leftarrow \text{compress}(0^{m+1} \parallel y_1) \)
- for i from 1 to k do
  - \( g_{i+1} \leftarrow \text{compress}(g_i \parallel 1 \parallel y_{i+1}) \)
- \( h(x) \leftarrow g_{k+1}; \ \text{return} \ (h(x)). \)
Theorem 4.6 (page 133)

- Theorem 4.6. If \texttt{compress}(x) is a collision resistant function, then \texttt{h}(x) in Algorithm 4.6 is a collision resistant hash function.

- Basic idea: If we can find a collision for \texttt{h}(x), we can find a collision for \texttt{compress}(x) in polynomial time.
Proof of Theorem 4.6

- Suppose $x \neq x'$ and $h(x) = h(x')$.
  - $y(x) = y_1 \| y_2 \| ... \| y_k \| y_{k+1}$
  - $y(x') = y'_1 \| y'_2 \| ... \| y'_l \| y'_{l+1}$

- Case 1: $|x| \neq |x'| \mod (t-1)$.
- Case 2a: $|x| = |x'|$
- Case 2b: $|x| \neq |x'|$ but $|x| = |x'| \mod (t-1)$
Case 1: $|x| \neq |x'| \mod (t-1)$

- This implies different number of 0’s padded $d \neq d'$ and $y_{k+1} \neq y'_{l+1}$.
  - $g_{k+1} = \text{compress}(g_k \| 1 \| y_{k+1}) = h(x)$
  - $h(x') = g'_{l+1} = \text{compress}(g'_{l} \| 1 \| y'_{l+1})$
  - Since $h(x) = h(x')$ but $y_{k+1} \neq y'_{l+1}$, we have found a collision (which one ?) for $\text{compress}(z)$. 
Case 2a: \(|x| = |x'|\)

- This means \(k = l\) and \(y_{k+1} = y'_{k+1}\).
- Similar to Case 1, we have
  - \(\text{compress}(g_k || 1 || y_{k+1}) = \text{compress}(g'_k || 1 || y'_{k+1})\)
  - if \(g_k \neq g'_k\), then we have found a collision for \(\text{compress()}\).
  - if \(g_k = g'_k\), use the same argument, find
    - \(\text{compress}(g_{k-1} || 1 || y_k) = \text{compress}(g'_{k-1} || 1 || y'_k)\)
    - if \(g_{k-1} \neq g'_{k-1}\) or \(y_k \neq y'_k\), then we have found a collision for \(\text{compress()}\) else repeat.
Case 2b:

$|x| \neq |x'|$ but $|x| = |x'| \mod(t-1)$

- Without loss of generality, assume $k < l$
- Applying similar to Case 2a, we have
  - either found the collision during the iteration process, or we reach the final situation that
  - $\text{compress}(0^{m+1} || y_1) = g_1 = g_{l-k+1} = \text{compress}(g'_{l-k} || 1 || y'_{l-k+1})$.
  - $m$-th bit of the $0^{m+1} || y_1$ is 0, which is different from the $m$-th bit of $g'_{l-k} || 1 || y'_{l-k+1}$. Hence, we have found a collision.
Algorithm 4.7. Merkle-Damgard(x), t = 1.

- Suppose that compress: \( \{0,1\}^{m+1} \rightarrow \{0,1\}^m \) is a collision-resistant function. \( f(0)=0; f(1)=01 \).
- \( n = |x|, x = x_1 \parallel x_2 \parallel ... \parallel x_n \)
- \( y \leftarrow 11 \parallel f(x_1) \parallel f(x_2) \parallel ... \parallel f(x_n) \)
- Let \( y = y_1 \parallel y_2 \parallel ... \parallel y_{k-1} \parallel y_k \)
- \( g_1 \leftarrow \text{compress}(0^m \parallel y_1) \)
- for i from 1 to k-1 do
  - \( g_{i+1} \leftarrow \text{compress}(g_i \parallel y_{i+1}) \)
- \( h(x) \leftarrow g_k \); return \( h(x) \).
Theorem 4.7 (page 135)

Theorem 4.7. If \texttt{compress}(x) is a collision resistant function, then \texttt{h(x)} in Algorithm 4.7 is a collision resistant hash function.

Basic idea: Similar to Theorem 4.6 and the property that there is no \( x \neq x' \) that \( y(x) \) and \( y(x') \) are prefix/postfix of each other.
Secure Hash Algorithm (SHA)

History of SHA:
- 1990: MD4 proposed by Rivest.
- 1993: SHA by NIST adopted as FIPS180.
- 1995: SHA-1, minor variation of SHA (renamed as SHA-0), as FIPS180-1.
Various SHA Properties

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message Size (bits)</th>
<th>Block Size (bits)</th>
<th>Word Size (bits)</th>
<th>Message Digest Size (bits)</th>
<th>Security² (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>$&lt; 2^{64}$</td>
<td>512</td>
<td>32</td>
<td>160</td>
<td>80</td>
</tr>
<tr>
<td>SHA-256</td>
<td>$&lt; 2^{64}$</td>
<td>512</td>
<td>32</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>SHA-384</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>SHA-512</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
</tbody>
</table>

Secure Hash Algorithm Properties
Cryptosystem 4.1 SHA-1(x)

- \( y \leftarrow SHA-1-PAD(x) = M_1 || M_2 \ldots || M_n \)
  - each \(|M_i| = 512\), \(M_i = (W_0, W_1, \ldots, W_{15})\).
- \((H_0, H_1, H_2, H_3, H_4) \leftarrow IV\) (160-bits)
- for \(i\) from 1 to \(n\) do
  - \((A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)\)
  - \((A, B, C, D, E) \leftarrow compress(A, B, C, D, E, M_i)\)
  - \((H_0, H_1, H_2, H_3, H_4) += (A, B, C, D, E) \mod 2^{32}\)
- return \((H_0 || H_1 || H_2 || H_3 || H_4)\)
SHA-1 Initial Vector (IV)

- For SHA-1, the initial hash value, $H_0$, $H_1$, $H_2$, $H_3$, $H_4$, consists of five 32-bit words, in hex.
- Total bits = 160.
- Q: How about replace IV with a private key?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_0$</td>
<td>67452301</td>
</tr>
<tr>
<td>$H_1$</td>
<td>EFCDAB89</td>
</tr>
<tr>
<td>$H_2$</td>
<td>98BADCFE</td>
</tr>
<tr>
<td>$H_3$</td>
<td>10325476</td>
</tr>
<tr>
<td>$H_4$</td>
<td>C3D2E1F0</td>
</tr>
</tbody>
</table>
Algorithm 4.8: SHA-1-PAD(x)

- Assume length of binary x, $|x| \leq 2^{64}-1$.
- $L = 64$-bit binary representation of $|x|$.
- $y \leftarrow x || 1 || 0^d || L$, where
  - $0^d = d$ copies of 0.
  - $d \leftarrow (447-|x|) \mod 512$. (why ?)
    - $|y|$ must be a multiple of 512.
    - $|x|+ 1 + d + 64 = 0 \mod 512$.
    - $d = -65 - |x| = (447- |x|) \mod 512$. 
32-bits Operations in SHA-1

<table>
<thead>
<tr>
<th>operations</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \land Y$</td>
<td>$X$ bitwise “and” $Y$</td>
</tr>
<tr>
<td>$X \lor Y$</td>
<td>$X$ bitwise “or” $Y$</td>
</tr>
<tr>
<td>$X \oplus Y$</td>
<td>$X$ bitwise “xor” $Y$</td>
</tr>
<tr>
<td>$X + Y$</td>
<td>$X + Y \mod 2^{32}$</td>
</tr>
<tr>
<td>$\sim X$</td>
<td>bitwise 1s complement of $X$</td>
</tr>
<tr>
<td>$\text{ROTL}^s(X)$</td>
<td>circular left shift of $X$ by $s$ bits.</td>
</tr>
</tbody>
</table>
compress(A,B,C,D,E,M_i)

- $|M_i| = 512$, $M_i = (W_0, W_1, ..., W_{15})$.
- for t from 16 to 79 do
  - $W_t \leftarrow \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
- for t from 0 to 79 do
  - $T \leftarrow \text{ROTL}^5(A) + f_t(B,C,D) + E + W_t + K_t$
  - $E \leftarrow D$; $D \leftarrow C$; $C \leftarrow \text{ROTL}^{30}(B)$; $B \leftarrow A$; $A \leftarrow T$
  - $f_t: \{0,1\}^{32 \times 3} \rightarrow \{0,1\}^{32}$. $K_t$: 32-bit key.
SHA-1 $f_t(B,C,D)$ functions

- $f_t(B,C,D) =$
  - $(B \land C) \lor ((\sim B) \land D)$, if $0 \leq t \leq 19$.
  - $B \oplus C \oplus D$, if $20 \leq t \leq 39$.
  - $(B \land C) \lor (B \land D) \lor (C \land D)$, if $40 \leq t \leq 59$.
  - $B \oplus C \oplus D$, if $60 \leq t \leq 79$. 
SHA-1 constants: $K_t$

- $K_t =$
  - 5A827999, if $0 \leq t \leq 19$.
  - 6ED9EBA1, if $20 \leq t \leq 39$.
  - 8F1BBCDC, if $40 \leq t \leq 59$.
  - CA62C1D6, if $60 \leq t \leq 79$. 
SHA-1 Example
(Short Message, Initial setup)

- message “abc” has length 24. Its binary representation is 00..011000.
- number of 0’s to be padded, \( d = 447 - 24 = 423 \). Hence, \( y \) is one block of 512 bits below:

\[
\begin{align*}
\text{“a”} & \quad 01100001 \\
\text{“b”} & \quad 01100010 \\
\text{“c”} & \quad 01100011 \quad 1 \quad 00...00 \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad 00...011000.
\end{align*}
\]
SHA-1 Example
(Short Message, 1\textsuperscript{st} Block)

<table>
<thead>
<tr>
<th>$W_0$</th>
<th>61626380</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_1$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_2$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_3$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_4$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_5$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_6$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_7$</td>
<td>00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$W_8$</th>
<th>00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_9$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{10}$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{11}$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{12}$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{13}$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{14}$</td>
<td>00000000</td>
</tr>
<tr>
<td>$W_{15}$</td>
<td>000000018</td>
</tr>
<tr>
<td>t</td>
<td>a</td>
</tr>
<tr>
<td>-----</td>
<td>--------------</td>
</tr>
<tr>
<td>0</td>
<td>0116fc33</td>
</tr>
<tr>
<td>1</td>
<td>8990536d</td>
</tr>
<tr>
<td>2</td>
<td>a1390f08</td>
</tr>
<tr>
<td>76</td>
<td>a079b7d9</td>
</tr>
<tr>
<td>77</td>
<td>860d21cc</td>
</tr>
<tr>
<td>78</td>
<td>5738d5e1</td>
</tr>
<tr>
<td>79</td>
<td>42541b35</td>
</tr>
</tbody>
</table>
SHA-1 Example (Final Output)

t = 79 : 42541b35  5738d5e1  21834873  681e6df6  d8fdf6ad

For SHA-1, the initial hash value, \(H^{(0)}\), is

\[
H_0^{(0)} = 67452301 \\
H_1^{(0)} = \text{efcdab89} \\
H_2^{(0)} = \text{98badcfe} \\
H_3^{(0)} = 10325476 \\
H_4^{(0)} = \text{c3d2e1f0}.
\]

\[
H_0^{(1)} = 67452301 + 42541b35 = \text{a9993e36} \\
H_1^{(1)} = \text{efcdab89} + 5738d5e1 = \text{4706816a} \\
H_2^{(1)} = \text{98badcfe} + 21834873 = \text{ba3e2571} \\
H_3^{(1)} = 10325476 + 681e6df6 = \text{7850c26c} \\
H_4^{(1)} = \text{c3d2e1f0} + \text{d8fdf6ad} = \text{9cd0d89d}.
\]
SHA-1 Example (Long Message)

- $x =$ binary-coded form of the ASCII string which consists of 1,000,000 repetitions of the character “a”.
- Find the values of the parameters involved (e.g. n, d, y, and last block):
  - “a” = 01100001$_2$.
  - $n = |x| =$ length of bit-string = 1,000,000*8 and its binary = 11110100001001001000000000.
SHA-1 Example
(Long Message, Parameters)

- \( n = 8 \times 1,000,000 / 512 = 15625 \) blocks.
- \( d = 512 - 64 - 1 - 0 = 447 \) 0’s padded.
  - (why ?)
- \( y = 1,000,000 \) copies of 01100001 ("a") followed by 1, padded with 447 0’s, and
  - \( L = 000\ldots01111010000100100000000000. \)
  - (L is left-padded with 0 to make 64 bits)
SHA-1 Example
(Long Message, Blocks Info.)

- First 15625 blocks: 64 copies of “a”.
- Last block: 1 || 0^{447} || L.
  - L=000…0111101000010010000000000.
- The resulting SHA-1 message digest is:
  - 34AA973C D4C4DAA4 F61EEB2B DBAD2731 6534016F.
Collisions for hash functions

- Mid-1990: collisions of compression function used in MD4 and MD5 found.
- 1998: collisions for SHA-0 can be found in $O(2^{61})$, better than birthday attack of $O(2^{80})$.
- 2004: actual collisions found for SHA-0, MD5.
- 2005: actual collisions found for (58-round) reduced version of SHA-1.
- 2005: estimate collisions for SHA-1 can be found in $O(2^{69})$. [Wang, Yin, and Yu]
Summary

- General step of iterated hash function.
- Merkle-Damgard Construction Method.
  - Use compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ a collision-resistant function to construct a “provable” collision-resistant hash function of a large text.
  - two cases: (a) $t \geq 2$, (b) $t \geq 1$.
- SHA-1, PAD function, and examples.