From compression to hash

Iterated hash functions

Comparison of security criteria

- In order for a hash function to be secure, three problems should be hard to solve: Preimage; Second Preimage; and Collision.
- Which is the hardest, and how do we tell?
**Collision ≤_T Second Preimage**

**Algorithm:** CollisionToSecondPreimage(h)

```plaintext
external Oracle2ndPreimage
choose x \in X uniformly at random
if (Oracle2ndPreimage(h, x)=x' and (x \neq x') and (h(x')=h(x))
    then return (x, x')
else return (failure)
```

*As a consequence, we say that the property of collision resistance implies the property of second preimage resistance.*

---

**Collision Avoidance**

- Can Collision be reduced to Preimage?
- If so, we need only avoid Collision to ensure against all three security problems.
Collision $\leq_T$ Preimage

Algorithm: CollisionToPreimage($h$)

```
external OraclePreimage
choose $x \in X$ uniformly at random
$y \in h(x)$
if (OraclePreimage($h, y$) = $x'$ and ($x \neq x'$))
    then return ($x, x'$)
else return (failure)
```

Proving Collision $\leq_T$ Preimage

**Theorem.** Suppose $h: X \to Y$ is a hash function where $|X|$ and $|Y|$ are finite and $|X| \geq 2|Y|$. Suppose OraclePreimage is a $(1, q)$ algorithm for Preimage, for the fixed hash function $h$. Then CollisionToPreimage is a $(1/2, q+1)$ algorithm for Collision, for the fixed hash function $h$.

**Proof.** We compute the average-case probably of success for CollisionToPreimage.
Equivalence relations

- For $x \in X$, define $x \sim x_1$ of $h(x) = h(x_1)$. We show that $\sim$ is an equivalence relation.
- Denote the equivalence class of $x$ under $\sim$ by $[x] = \{x_1 \in X : x \sim x_1\}$.
- Note that $|[x]|$ is the number of possible $x_1$ that could be returned by OraclePreimage and $|[x]|^{-1}$ is the number different from $x$.

Average-case probability of success

$$\mathbb{P}[\text{success}] = \frac{1}{|X|} \prod_{x \in X} \frac{|[x]|!}{|[x]|}$$
$$= \frac{1}{|X|} \prod_{c \in C} \frac{|[c]|!}{|[c]|}$$
$$= \frac{1}{|X|} \prod_{c \in C} |C|!$$
$$\geq \frac{|X|! |Y|}{|X|!}$$
$$\geq \frac{1}{2}$$

Since $h$ is surjective, there is one equivalence class for each element of $Y$. 
Iterated hash functions

- We study a particular method for extending compression functions to hash functions with an infinite domain.

- Suppose

\[ \text{compress} : \{0,1\}^{m+t} \to \{0,1\}^m \]

is a compression function where \( t \geq 1 \).

Three main steps of an iterated hash

Preprocessing

Given input string \( x \), where \( |x| \geq m+t+1 \), construct string

\[ y = y_1 \ || \ y_2 \ || \ldots \ || \ y_r, \]

where \( |y_i| = t \) for \( 1 \leq i \leq r \) using a public algorithm.

Processing

Let IV be a public initial bitstring of length \( m \). Compute

\[ z_0 \ \Box \ IV \]

\[ z_1 \ \Box \ \text{compress}(z_0 \ || \ y_1) \]

\[ \ldots \]

\[ z_r \ \Box \ \text{compress}(z_{r-1} \ || \ y_r). \]

Optional output transformation

Define \( h(x) = g(z_r) \), where \( g : \{0,1\}^m \to \{0,1\}^l \) is a public function.
The Merkle-Damgard construction

- Suppose \( \text{compress} : \{0,1\}^{m+t} \rightarrow \{0,1\}^{m} \) is a collision resistant hash function. We use \( \text{compress} \) to construct a collision resistant hash function \( h : \times \rightarrow \{0,1\}^{m} \), where elements of \( \times \) are bit strings of length \( m+t+1 \) or larger.

- Express \( x \in \times \) as the concatenation
  \[
  x = x_1 || x_2 || \ldots || x_k,
  \]
  where
  \[
  |x_1| = |x_2| = \ldots = |x_{k-1}| = t-1
  \]
  and
  \[
  |x_k| = t - 1 - d
  \]
  with \( 0 \leq d \leq t-2 \).

Merkle-Damgard

Algorithm Merkle-DamGard\((x)\)

\begin{verbatim}
external compress
comment: compress : \{0,1\}^{m+t} \rightarrow \{0,1\}^{m}, where t \geq 2
n \in |x|
k \in ceiling[n/t-1]]
d \in n - k(t - 1)
for i \in 1 to k - 1
  do y_i \in x_i
    y_{k-1} \in binary representation of d
    z_i \in 0^{m-1} || y_i
    g_i \in compress(z_i)
    for i \in 1 to k
      do z_{i+1} \in g_i || 1 || y_{i+1}
          g_{i+1} \in compress(z_{i+1})
h(x) \in g_{k+1}
return (h(x))
\end{verbatim}
The resulting hash is collision resistant

**Theorem.** Suppose compress : \( \{0,1\}^{m+t} \rightarrow \{0,1\}^m \) is a collision resistant function, where \( t \geq 2 \). Then the function

\[
h : \bigcup_{i=m+1}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^m
\]

as constructed in the Merkle-Damgard algorithm is a collision resistant hash function.

**Proof.** Suppose that we can find \( x \neq x' \) such that \( h(x) = h(x') \). We find a collision for compress in polynomial time.

**Case 1: \(|x| \neq |x'| \pmod{t-1}\)**

In this case, the remainders after division by \( t-1 \) are not equal, that is \( d \neq d' \). We have

\[
\text{compress}(g_k \ || \ 1 \ || \ y_{k+1}) = g_{k+1} = h(x) = h(x') = g'_{l+1} = \text{compress}(g'_k \ || \ 1 \ || \ y'_{l+1}),
\]

which is a collision for \( h \) because \( y_{k+1} \neq y'_{l+1} \).
Case 2a: $|x| = |x'|$

In this case, we have $k = l$ and $y_{k+1} = y'_{k+1}$

$$\text{compress}(g_k || 1 || y_{k+1}) = g_{k+1} = h(x) = h(x') = g'_{k+1} = \text{compress}(g'_{k} || 1 || y'_{k+1}).$$

If $g_k \neq g'_k$, then we find a collision for compress, so assume $g_k = g'_k$. Then we have

$$\text{compress}(g_{k-1} || 1 || y_k) = g_k = g'_k = \text{compress}(g'_{k-1} || 1 || y'_{k}).$$

Assuming no collisions, . . .

We continue working backwards, until

$$\text{compress}(0^{m-1} \ || \ y_1) = g_1 = g'_1 = \text{compress}(0^{m-1} \ || \ y'_1).$$

If $y_1 \neq y'_1$, then we find a collision for compress. Otherwise, $y_i = y'_i$ for $1 \leq i \leq k+1$, so $y(x) = y(x')$. This too is a problem. Why?
Case 2b: $|x| = |x'| \pmod{t-1}$ but $|x| \neq |x'|$

WLOG $|x| > |x'|$, so $\ell \geq k$. Arguing as in case 2a, we obtain
\[
\begin{align*}
\text{compress}(0^{m-1} \ || \ y_1) &= g_1 \\
&= g_{\ell-k+1}' \\
&= \text{compress}(g_{\ell-k}' \ || \ 1 \ || \ y_{\ell-k+1}).
\end{align*}
\]

But the $(m+1)$st bits of $0^{m-1} \ || \ y_1$ and $g_{\ell-k}' \ || \ 1 \ || \ y_{\ell-k+1}$ are different. Collision time.

Done!

The Secure Hash Algorithm

- SHA-1 is the current iterated hash of choice.*
- SHA-1 requires $|x| \leq 2^{64}-1$. The binary representation of $|x|$ is padded on the left with zeroes to that its length is exactly 64 bits.

*MD4, MD5 and SHA are ancestors, SHA-256, SHA-384 and SHA-512 soon to be descendants.
SHA-1 padding is easy

Algorithm: SHA-1-Pad(x)

comment: |x| ≤ 2^{64} - 1
\[ d = (447 - |x|) \mod 512 \]
\[ \ell \] the binary representation of |x|, where |\ell| = 64
\[ y = x || 1 || 0^d || \ell \]

SHA-1 is not!

- Define functions

\[ f_t(B,C,D) = \begin{cases} 
(B \oplus C) \quad (\Box B) \Box D & \text{if } 0 \leq t \leq 19 \\
B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\
(B \Box C) \quad (B \Box D) \quad (C \Box D) & \text{if } 40 \leq t \leq 59 \\
B \oplus C \oplus D & \text{if } 60 \leq t \leq 79 
\end{cases} \]

- Define constants

\[ K_t = \begin{cases} 
5A82799 & \text{if } 0 \leq t \leq 19 \\
6ED9BA1 & \text{if } 20 \leq t \leq 39 \\
8F1BCDC & \text{if } 40 \leq t \leq 59 \\
CA62C1D6 & \text{if } 60 \leq t \leq 79 
\end{cases} \]
The SHA-1 Cryptosystem

Algorithm 4.1: SHA-1(x)

external SHA-1-out

global K0, ..., K15

y := SHA-1-red(x)

define y = M1 || M2 || ... || Mn, where each Mi is a 512-bit block

H0 := 67452301
H1 := EFCDAB89
H2 := 98BADCFE
H3 := 10325476
H4 := C3D2E1F0

for i := 1 to n do:

    define Mi = W0 || W1 || ... || W15, where each Wi is a word

    for t := 16 to 79 do:

        Wi := ROTL(Wi-3 || Wi-5 || Wi-16 || Wi-16) + W11

        A := Ht
        B := Ht
        C := Ht
        D := Ht
        E := Ht

        for t := 0 to 79 do:

            tempr := ROTL7(A) + f(B, C, D) + E + Wi + Kt

            E := D
            D := C + ROTL30(B)
            C := B + A
            A := temp

        Ht := Ht + A
        Ht := Ht + B
        Ht := Ht + C
        Ht := Ht + D
        Ht := Ht + E

    return (H0 || H1 || H2 || H3 || H4)