Collision Resistant Hash functions and MACs

Integrity vs authentication

• **Message integrity** is the property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.

• **Message origin authentication** is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some time in the past.

• **Data origin authentication includes data integrity and vice versa**
Collision Resistant Hash Functions

CRHF & MACs

Hash functions: informal properties

- Informal properties
  - "easy" to compute
  - "unique"
  - "difficult" to invert

- The hash of a message can be used to "uniquely" represent the message
An example

Nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che' la diritta via era smarrita.

Ahi quanto a dir qual era e` cosa dura
esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!

MD5

d94f329333386d5abef6475313755e94
128 bit The hash size is fixed, generally smaller than the message size

Protecting files using C.R. hash

• Software packages

read-only public space
H(F₁)    H(F₂)    H(Fᵟ)

• When user downloads package, can verify that contents are valid
  – H collision resistant ⇒ attacker cannot modify package without detection

• no key needed (public verifiability), but requires read-only space
Properties: collisions

- A hash function $H: \{0,1\}^* \rightarrow \{0,1\}^m$
- Properties:
  - **Compression** – $H$ maps an input $x$ of arbitrary finite length into an output $H(x)$ of fixed length $m$
  - **Ease of computation** – given $x$, $h(x)$ must be “easy” to compute
- A hash function is many-to-one and thus implies **collisions**
  - A **collision** for $H$ is a pair $x_0, x_1$ s.t. $H(x_0) = H(x_1)$ and $x_0 \neq x_1$

Security properties

- **Preimage resistance (one-way)** – for essentially all pre-specified outputs, it is *computationally infeasible* to find any input which hashes to that output
  - i.e., to find $x$ such that $y = h(x)$ given $y$ for which $x$ is not known
- **2nd-preimage resistance (weak collision resistance)** – it is computationally infeasible to find any second input which has the same output as any specified input
  - i.e., given $x$, to find $x' \neq x$ such that $h(x) = h(x')$
- **Collision resistance (strong collision resistance)** – it is computationally infeasible to find any two distinct inputs which hash to the same output,
  - i.e., find $x, x'$ such that $h(x) = h(x')$
Classification

• A one-way hash function (OWHF) provides preimage resistance, 2-nd preimage resistance
  – OWHF is also called weak one-way hash function

• A collision resistant hash function (CRHF) provides 2-nd preimage resistance, collision resistance
  – CRHF is also called strong one-way hash function

Relationship between security properties

• Collision resistance implies 2-nd preimage resistance

• Collision resistance does not imply preimage resistance
  – In practice, CRHF almost always has the additional property of preimage resistance
Attacking Hash Function

• An attack is successful if it produces a collision

• Selective forgery: the adversary has complete, or partial, control over x

• Existential forgery: the adversary has no control over x

Black box attacks

• Black box attacks
  – Consider H as a black box
  – Only consider the output bit length $m$;
  – H approximates a random variable

• Specific BB attacks
  – Guessing attack: find a 2\textsuperscript{nd} pre-image (O(2\textsuperscript{m}))
  – Birthday attack: find a collision (O(2\textsuperscript{m/2}))

• These attacks constitute a security upper bound
Guessing attack

- Objective: to find a 2\textsuperscript{nd} pre-image
  - Given $x_0$, find $x_1 \neq x_0$ s.t. $H(x_0) = H(x_1)$

- Complexity
  - Every step requires
    - 1 random number generation: efficient!
    - 1 hash function computation: efficient!
  - Constant and negligible data/storage complexity
  - Time complexity: $2^m$

```python
def GuessingAttack(x_0):
    repeat
        x ← random(); // guessing
    until h(x_0) = h(x)
    return x
```

Birthday attack

- Algorithm
  1. Choose $N = 2^{n/2}$ random input messages $x_1, x_2, \ldots, x_N$ (distinct w.h.p.)
  2. For $i := 1$ to $N$ compute $t_i = H(x_i)$
  3. Look for a collision ($t_i = t_j$), $i \neq j$. If not found, go to step 1.

  - Running Time: $2^{n/2}$
  - Space: $2^{n/2}$
Birthday paradox

- Problem 1. In a room of 23 people, the probability that at least a person is born on 25 December is $23/365 = 0.063$

- Problem 2. In a room of 23 people, the probability that at least 2 people have the same birthdate is $0.507$

Let $r_1, \ldots, r_n \in \{1,\ldots,B\}$ be independent and identically distributed integers.

**Theorem**: when $n = 1.2 \times B^{1/2}$ then

$$\Pr[ \exists i \neq j:\ r_i = r_j ] \geq \frac{1}{2}$$
Sample hash functions

<table>
<thead>
<tr>
<th>Hash Function</th>
<th>$m$</th>
<th>Preimage</th>
<th>Collision</th>
<th>Speed (Mb/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>128</td>
<td>$2^{128}$</td>
<td>$2^{64}$</td>
<td></td>
</tr>
<tr>
<td>RIPEMD-128</td>
<td>128</td>
<td>$2^{128}$</td>
<td>$2^{64}$</td>
<td></td>
</tr>
<tr>
<td>SHA-1, RIPEMD-160</td>
<td>160</td>
<td>$2^{160}$</td>
<td>$2^{80}$</td>
<td>153</td>
</tr>
<tr>
<td>SHA-256</td>
<td>256</td>
<td>$2^{128}$</td>
<td>$2^{128}$</td>
<td>111</td>
</tr>
<tr>
<td>SHA-512</td>
<td>512</td>
<td>$2^{256}$</td>
<td></td>
<td>99</td>
</tr>
</tbody>
</table>
Use of CRHF

- The purpose of a CRHF, in conjunction with other mechanisms (authentic channel, encryption, digital signature), is to provide message integrity

Integrity with CRHF

CRHF and an authentic channel

- physically authentic channel
- digital signature
Integrity with CRHF

**CRHF and encryption**

- \( E(e, x||H(x)) \)
  - Confidentiality and integrity
  - As secure as \( E \)
- \( x, E(e, H(x)) \)
  - Sender has seen \( h(x) \)
- \( E(e, x), H(x) \)
  - \( H(x) \) can be used to check a guessed \( x \)

**How to build a CRHF**

- **Goal**: Build a CRHF
- **Approach**: given a CRHF for short messages, construct a CRHF for long messages
- **Solution**: the Merkle-Damgard iterated construction
The Merkle-Damgard iterated construction

Given \( h: T \times X \rightarrow T \) (compression function) we obtain \( H: X^{\text{SL}} \rightarrow T \). \( H_i \) - chaining variables

PB: padding block

1000...0 \( \| \) msg len

64 bits

If no space for PB add another block

M-D collision resistance

- **Theorem.** if \( h \) is collision resistant then so is \( H \).
- **Proof:** collision on \( H \) \( \Rightarrow \) collision on \( h \)

- To construct a CRHF, it suffices to construct a collision resistant compression function
Compression function

- Block cipher
- **Davies-Meyer compression function**
  - Finding a collision $h(H, m) = h(H', m')$ requires $2^{n/2}$ evaluations of $(E, D)$ ⇒ best possible!

Message Authentication Code (MAC)

CRHF & MACs
Secure MACs

- **Ease of computation**
  - Given a function $S$, a key $k$ and an input $x$, $S(k, x)$ is easy to compute

- **Compression**
  - $S$ maps an input $x$ of arbitrary finite bitlength into an output of fixed length $m$

- **Computation-resistance**
  - For each key $k$, given zero or more $(x_i, t_i)$ pairs, where $t_i = S(e, x_i)$ (chosen message attack)
    
    it is *computationally infeasible* to compute $(x, t), t = S(k, x)$, for any new input $x \neq x_i$ (including possible $t = t_i$ for some $i$) (existential forgery)
Secure MACs: facts

- Attacker cannot produce a valid tag for any new message
  - Given \((m, t)\), attacker cannot even produce \((m, t')\) for \(t' \neq t\)
- Computation resistance implies key non-recovery (but not vice versa)
- For an adversary not knowing \(k\)
  - \(S\) must be 2nd-preimage and collision resistant;
  - \(S\) must be preimage resistant w.r.t. a chosen-text attack;
- Secure MAC definition says nothing about preimage and 2nd-preimage for parties knowing \(k\)

Combining MAC and ENC

- PT message: \(m\); transmitted message: \(m'\);
  encryption key: \(e\); MAC key: \(a\)
- Option 1 (SSL)
  - \(t = S(a, m)\); \(c = E(e, m || t)\), \(m' = c\)
- Option 2 (IpSec)
  - \(c = E(e, m)\); \(t = S(a, c)\); \(m' = c || t\)
- Option 3 (SSH)
  - \(c = E(e, m)\); \(t = S(a, m)\); \(m' = c || t\)
How to build a MAC

• From a PRF
  – CBC-MAC
  – NMAC
  – PMAC

• From a CRHF
  – HMAC

MAC from PRF

• THM. If \( F: K \times X \rightarrow Y \) is a secure PRF and \( 1/|Y| \) is negligible, then \( F \) defines a secure MAC

• \( |Y| \) must be large, say \( |Y| \geq 2^{80} \)

• AES is a MAC for 16-byte messages (small-MAC)

• How to convert a small-MAC into a large-MAC?
  – CBC-MAC (banking – ANSI X9.9, X9.19, FIPS 186-3)
  – HMAC (Internet protocols: SSL, IpSec, SSH)
Truncating MAC based on PRF

• THM. Let $F : K \times X \rightarrow \{0,1\}^m$ is a secure PRF
  the so is $F_w(k,m) = F(k, m)[1..w] \quad \forall 1\leq w\leq m$
  – If $S$ is a MAC based on a PRF outputting m-bit tags
  then the truncated MAC outputting w-bit, $w\leq m$, is
  secure... as long as $1/2^w$ is still negligible (say $w\geq64$)

CBC-MAC construction

CBC-MAC takes
• two independent keys $K$ and $K_1$
• an arbitrary # of input blocks
• PRF is a cipher

Without the last encryption, rawCBC would be insecure
Security bounds

• How many msgs can I CBC-MAC using the same key?
  – Let $q = \#\text{msgs CBC-MAC-ed with the same key } k$
  – It can be proven that after $q$ msgs, the probability $P$ that MAC becomes insecure is $q^2/|X|$
    • AES: $|X| = 2^{128}$ and $P < 1/2^{32} \Rightarrow q < 2^{48}$ (GOOD!)
    • 3DES: $|X| = 2^{64}$, $P < 1/2^{32} \Rightarrow q < 2^{16}$ (BAD!)

MAC Padding

• Pad by zeroes $\Rightarrow$ insecure
  – pad$(m)$ and pad$(m||0)$ have the same MAC

• Padding must be an invertible function
  – $m_0 \neq m_1 \Rightarrow \text{pad}(m_0) \neq \text{pad}(m_1)$

• Standard padding (ISO)
  – Append “100…00” as needed
    • Scan right to left
    • “1” determines the beginning of the pad
  – Add a dummy block if necessary
    • When the message is a multiple of the block
    • The dummy block is necessary or existential forgery arises
CMAC

- CMAC uses $k_1$ and $k_2$ derived from $k$
- We don’t need the final encryption anymore

\[ F(k,..) \]

HMAC

Can we use a CRHF to build a MAC?

- $S(k, m) = H(k||m)$ is insecure!
HASH – insecure scheme

- Let \((m, t), \text{ where } t = S(k, m)\)
- It is “easy” to build a pair \((m', t'), \text{ where } t' = S(k, m')\)
  - Let \(m' = m || PB || w\), where \(w\) is a block, then
  - \(t' = h(w, t) \Rightarrow \text{existential forgery}\)

HMAC

Standard

- HMAC: \(S(k, m) = H(k \oplus \text{opad} || H(k \oplus \text{ipad} || p))\)
  - \text{ipad} and \text{opad} are fixed and predefined
  - Standard uses SHA-256 (PRF)
  - TLS: HMAC-SHA1-96
    - SHA1 is not collision resistant but HMAC needs only that the compression function is a PRF
  - \textbf{Security bounds.}
    \(\Pr [\text{after } q \text{ MACs, HMAC becomes insecure}] = q^2/|T|\)
    - SHA-256: \(q << 2^{128}\) (GOOD!)
HMAC – secure scheme

Timing Attack

- Example: Keyczar crypto library (Python) [simplified]

  ```python
def Verify(key, msg, sig_bytes):
    return HMAC(key, msg) == sig_bytes
```

- The problem: ‘==‘ implemented as a byte-by-byte comparison
- Comparator returns false when first inequality found
Timing attack

Timing attack: to compute tag for target message do:
Step 1: Query server with random tag
Step 2: Loop over all possible first bytes and query server. stop when verification takes a little longer than in step 1
Step 3: repeat for all tag bytes until valid tag found

Defense #1

Make string comparator always take same time (Python):

```python
return false if sig_bytes has wrong length
result = 0
for x, y in zip( HMAC(key, msg) , sig_bytes):
    result |= ord(x) ^ ord(y)
return result == 0
```

Can be difficult to ensure due to optimizing compiler
Defense #2

Make string comparator always take same time (Python):

```python
def Verify(key, msg, sig_bytes):
    mac = HMAC(key, msg)
    return HMAC(key, mac) == HMAC(key, sig_bytes)
```

*Attacker doesn’t know values being compared!*