CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography
[MACs and Hash Functions]

Spring 2019

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
• Reminders:
  – Lab 1 due on Monday
  – My office hours tomorrow @11am (not 9am)

• Coming up:
  – Guest lecture on Monday
    • Ivan Evtimov (UW) on adversarial ML
  – Homework 2 on crypto
    • Out by Monday, due Friday May 10
  – Web security starting mid next week
Recap: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.

Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.
Reminder: CBC Mode Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity
CBC-MAC

- Not secure when system may MAC messages of different lengths.
- NIST recommends a derivative called CMAC [FYI only]
Another Tool: Hash Functions
You Just Did This

```
franzi@codered:~/sploits$ md5sum sploit0.c
3a2e6ce795bce4d06df1ff6835d25cea  sploit0.c
franzi@codered:~/sploits$
```
Hash Functions: Main Idea

- Hash function $H$ is a lossy compression function
  - Collision: $h(x) = h(x')$ for distinct inputs $x$, $x'$
- $H(x)$ should look “random”
  - Every bit (almost) equally likely to be 0 or 1
- Cryptographic hash function needs a few properties...
Property 1: One-Way

• Intuition: hash should be hard to invert
  – “Preimage resistance”
  – Let $h(x') = y \in \{0,1\}^n$ for a random $x'$
  – Given $y$, it should be hard to find any $x$ such that $h(x)=y$

• How hard?
  – Brute-force: try every possible $x$, see if $h(x)=y$
  – SHA-1 (common hash function) has 160-bit output
    • Expect to try $2^{159}$ inputs before finding one that hashes to $y$. 
Property 2: Collision Resistance

• Should be hard to find $x \neq x'$ such that $h(x) = h(x')$
Birthday Paradox

• Are there two people in the first 1/6 of this classroom that have the same birthday?
  – 365 days in a year (366 some years)
    • Pick one person. To find another person with same birthday would take on the order of \( \frac{365}{2} = 182.5 \) people
    • Expect birthday “collision” with a room of only 23 people.
    • For simplicity, approximate when we expect a collision as \( \sqrt{365} \).

• Why is this important for cryptography?
  – \( 2^{128} \) different 128-bit values
    • Pick one value at random. To exhaustively search for this value requires trying on average \( 2^{127} \) values.
    • Expect “collision” after selecting approximately \( 2^{64} \) random values.
    • 64 bits of security against collision attacks, not 128 bits.
Property 2: Collision Resistance

• Should be hard to find \( x \neq x' \) such that \( h(x) = h(x') \)

• Birthday paradox (informal)
  – Let \( t \) be the number of values \( x, x', x'' \ldots \) we need to look at before finding the first pair \( x, x' \) s.t. \( h(x) = h(x') \)
  – What is probability of collision for each pair \( x, x' \)? \( \frac{1}{2^n} \)
  – How many pairs \( x, x' \) would we need to look at before finding the first collision? \( O(2^n) \)
  – How many pairs \( x, x' \) total? \( \text{Choose}(t, 2) = \frac{t(t-1)}{2} \sim O(t^2) \)
  – What is \( t \), the number of values we need to look at? \( 2^{n/2} \)

• Brute-force collision search is only \( O(2^{n/2}) \), not \( O(2^n) \)
  – For SHA-1, this means \( O(2^{80}) \) vs. \( O(2^{160}) \)
Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x) = h(x')$
- Birthday paradox means that brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
One-Way vs. Collision Resistance

• One-wayness does not imply collision resistance
  – Suppose $g$ is one-way
  – Define $h(x)$ as $g(x')$ where $x'$ is $x$ except the last bit
    • $h$ is one-way (to invert $h$, must invert $g$)
    • Collisions for $h$ are easy to find: for any $x$, $h(x_0)=h(x_1)$

• Collision resistance does not imply one-wayness
  – Suppose $g$ is collision-resistant
  – Define $y=h(x)$ to be $0x$ if $x$ is $n$-bit long, $1g(x)$ otherwise
    • Collisions for $h$ are hard to find: if $y$ starts with $0$, then there are no collisions, if $y$ starts with $1$, then must find collisions in $g$
    • $h$ is not one way: half of all $y$’s (those whose first bit is $0$) are easy to invert (how?); random $y$ is invertible with probab. $\frac{1}{2}$
Property 3: Weak Collision Resistance

• Given randomly chosen x, hard to find x’ such that h(x)=h(x’)
  – Attacker must find collision for a specific x. By contrast, to break collision resistance it is enough to find any collision.
  – Brute-force attack requires $O(2^n)$ time

• Weak collision resistance does not imply collision resistance.
Hashing vs. Encryption

• Hashing is one-way. There is no “un-hashing”
  – A ciphertext can be decrypted with a decryption key...
    hashes have no equivalent of “decryption”
• Hash(x) looks “random” but can be compared for equality with Hash(x’)
  – Hash the same input twice \(\rightarrow\) same hash value
  – Encrypt the same input twice \(\rightarrow\) different ciphertexts
• Cryptographic hashes are also known as “cryptographic checksums” or “message digests”
Application: Password Hashing

• Instead of user password, store \( \text{hash}(\text{password}) \)
• When user enters a password, compute its hash and compare with the entry in the password file
• Why is hashing better than encryption here?

• System does not store actual passwords!
• Cannot go from hash to password!
**Application: Software Integrity**

**Goal:** Software manufacturer wants to ensure file is received by users without modification.

**Idea:** given `goodFile` and `hash(goodFile)`, very hard to find `badFile` such that `hash(goodFile)=hash(badFile)`
Which Property Do We Need?

• UNIX passwords stored as hash(password)
  – One-wayness: hard to recover the/a valid password
• Integrity of software distribution
  – Weak collision resistance
  – But software images are not really random... may need full collision resistance if considering malicious developers
Which Property Do We Need?

• UNIX passwords stored as hash(password)
  – **One-wayness**: hard to recover the/a valid password

• Integrity of software distribution
  – **Weak collision resistance**
  – But software images are not really random... may need **full collision resistance** if considering malicious developers

• Private auction bidding
  – Alice wants to bid B, sends H(B), later reveals B
  – **One-wayness**: rival bidders should not recover B (this may mean that she needs to hash some randomness with B too)
  – **Collision resistance**: Alice should not be able to change her mind to bid B’ such that H(B)=H(B’)

4/24/19

CSE 484 / CSE M 584 - Spring 2019
Common Hash Functions

• MD5 – Don’t Use!
  – 128-bit output
  – Designed by Ron Rivest, used very widely
  – Collision-resistance broken (summer of 2004)

• RIPEMD-160
  – 160-bit variant of MD5

• SHA-1 (Secure Hash Algorithm)
  – 160-bit output
  – US government (NIST) standard as of 1993-95
  – Theoretically broken 2005; practical attack 2017!

• SHA-256, SHA-512, SHA-224, SHA-384

• SHA-3: standard released by NIST in August 2015
SHA-1 Broken in Practice (2017)

Google just cracked one of the building blocks of web encryption (but don’t worry)

It’s all over for SHA-1

by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST

https://shattered.io
Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.

**Integrity and authentication:** only someone who knows KEY can compute correct MAC for a given message.
HMAC

• Construct MAC from a cryptographic hash function
  – Invented by Bellare, Canetti, and Krawczyk (1996)
  – Used in SSL/TLS, mandatory for IPsec

• Why not encryption?
  – Hashing is faster than block ciphers in software
  – Can easily replace one hash function with another
  – There used to be US export restrictions on encryption
Authenticated Encryption

• What if we want both privacy and integrity?
• Natural approach: combine encryption scheme and a MAC.
• But be careful!
  – Obvious approach: Encrypt-and-MAC
  – Problem: MAC is deterministic! same plaintext → same MAC
Authenticated Encryption

• Instead: Encrypt \textit{then} MAC.

• (Not as good: MAC-then-Encrypt)