CS 166: *Information Security*

Cryptographic Hash Functions

Prof. Tom Austin
San José State University

Cryptographic Hash Functions

or, *Why can't they tell me my password?*

Cryptographic hash functions

Encrypt data so that it can never be decrypted. Why is this useful?

- Efficient signatures
- Safely storing passwords
- "Proof of work" protocols

Hash functions in action

h("secret") = 5ebe2294ecd0e0f08eab7690d2a6ee69

Username	**PasswordHash**
Alice	5ebe2294ecd0e0f08eab7690d2a6ee69
Bob	4bbfbb9beab959cc431ec4eed504cde5
Charlie	5f202e7ab75f00af194c61cc07ae6b0c
David	3feb2d8fe13b4e9c3c81de0734257103

Hash and Sign

- Suppose Alice signs M
  - Alice sends M and S = $[M]_{Alice}$ to Bob
  - Bob verifies that M = $\{S\}_{Alice}$
  - Can Alice just send S?
- If M is big, $[M]_{Alice}$ costly to ***compute & send***
- Suppose Alice signs h(M) instead, where h(M) is much smaller than M.
  - Alice sends M and S = $[h(M)]_{Alice}$ to Bob
  - Bob verifies that h(M) = $\{S\}_{Alice}$

Hash and Sign Collision

- So, Alice signs h(M)
  - That is, Alice computes $S = [h(M)]_{Alice}$
  - Alice then sends (M, S) to Bob
  - Bob verifies that $h(M) = \{S\}_{Alice}$
- What if Trudy finds M’ so that h(M) = h(M’)
  - Then Trudy can replace (M, S) with (M’, S)
  - Bob does not detect tampering, since $h(M') = h(M) = \{S\}_{Alice}$
- What properties must h(M) satisfy?

Crypto Hash Function Properties

- Crypto hash function h(x) must provide
  - **Compression** – output length is small
  - **Efficiency** – h(x) easy to compute for any x
  - **One-way** – given a value y it is infeasible to find an x such that h(x) = y
  - **Weak collision resistance** – given x and h(x), infeasible to find y ≠ x such that h(y) = h(x)
  - **Strong collision resistance** – infeasible to find ***any*** x and y, with x ≠ y such that h(x) = h(y)
- Lots of collisions exist, but hard to find ***any***

Pre-Birthday Problem

- Suppose N people in a room
- How large must N be before the probability someone has same birthday as me is ≥ 1/2 ?
  - Solve: $1/2 = 1 - (364/365)^N$ for N
  - We find $N = 253$

Birthday Problem

- How many must be in a room before prob. is **≥ 1/2** that *any* two have same birthday?
  - 1 - 365/365 · 364/365 · · ·(365-N+1)/365
  - Set equal to 1/2 and solve: **N = 23**
- Surprising? A paradox?
- Maybe not: "Should be" about sqrt(365) since we compare all **pairs** x and y
  - And there are 365 possible birthdays

Of Hashes and Birthdays

- If h(x) is N bits, 2^N different hash values are possible
- So, if you hash about $2^{N/2}$ random values then you expect to find a collision
  - Since $\text{sqrt}(2^N) = 2^{N/2}$
- **Implication:** easier to brute-force hashes
  - secure N bit symmetric key requires 2^{N-1} work
  - secure N bit hash requires $2^{N/2}$ work

Non-crypto Hash (1)

- Data $X = (X_0,X_1,X_2,\ldots,X_{n-1})$, each X_i is a byte
- Define $h(X) = X_0+X_1+X_2+\ldots+X_{n-1}$
- Is this a secure cryptographic hash?
- Example: X = (10101010, 00001111)
- Hash is h(X) = 10111001
- If Y = (00001111, 10101010) then h(X) = h(Y)
- Easy to find collisions, so **not** secure…

Non-crypto Hash (2)

- Data $X = (X_0,X_1,X_2,\ldots,X_{n-1})$
- Suppose hash is defined as

  $h(X) = nX_0+(n-1)X_1+(n-2)X_2+\ldots+1\cdot X_{n-1}$
- Is this a secure cryptographic hash?
- Note that

  $h(10101010, 00001111) \neq h(00001111, 10101010)$
- But hash of (00000001, 00001111) is same as hash of (00000000, 00010001)
- Not "secure", but this hash is used in the (non-crypto) application rsync

Non-crypto Hash (3)

- Cyclic Redundancy Check (CRC)
- Essentially, CRC is the remainder in a long division calculation
- Good for detecting burst **errors**
  - Random errors unlikely to yield a collision
- But easy to construct collisions
- CRC has been mistakenly used where crypto integrity check is required (e.g., WEP)

Avalanche Effect

- Desired property: **avalanche effect**
  - Change to 1 bit of input should affect about half of output bits
- Crypto hash functions consist of some number of rounds
- Want security and speed
  - Avalanche effect after few rounds
  - But simple rounds
- Analogous to design of block ciphers

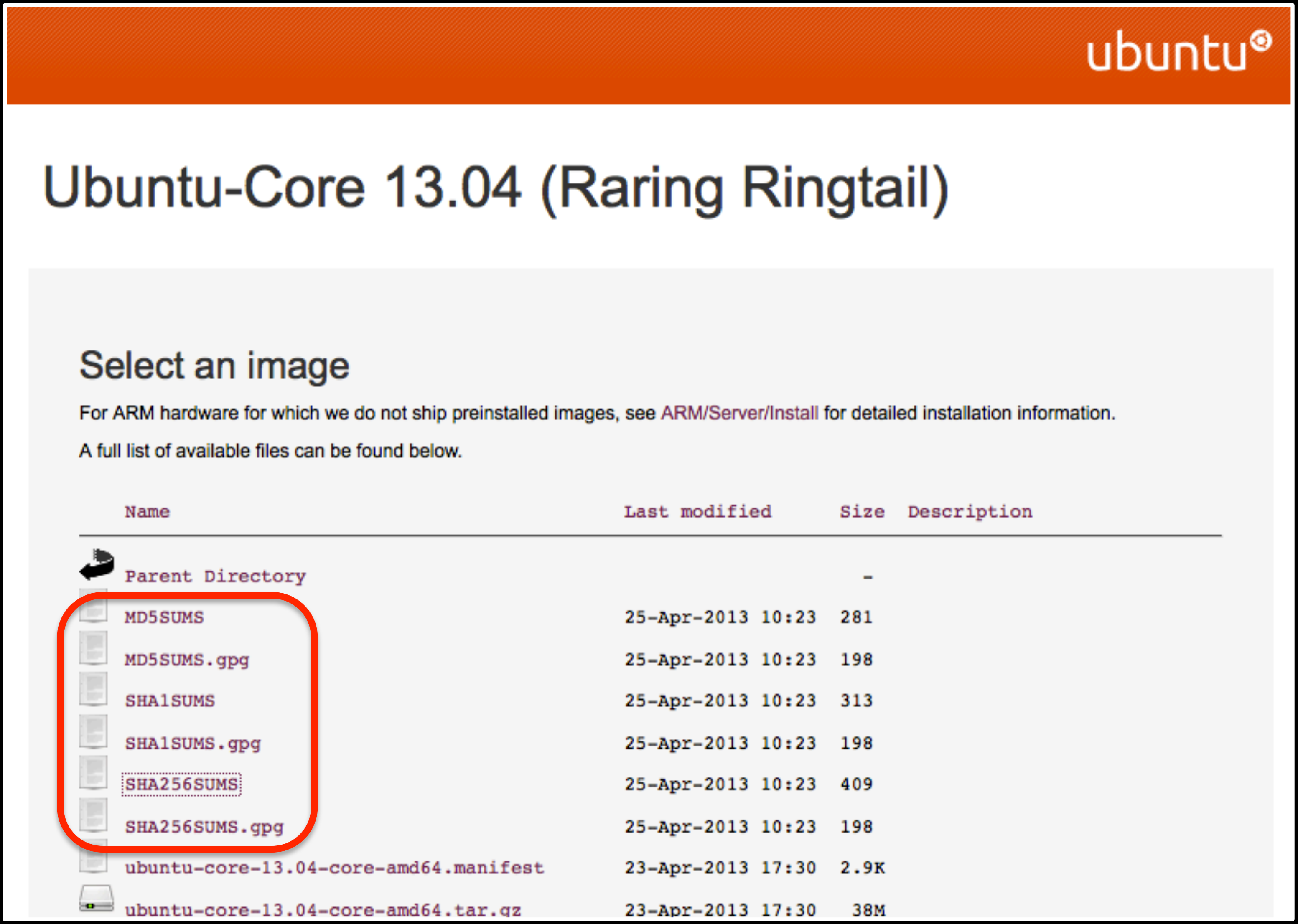
Avalanche Effect

```
Tiger("better call saul") =
   0201b60356a7eca259ff4d71
   ea910b83a316ceaed29f9d0a

Tiger("better call paul") =
   a9c6722a7a338cb292787d74
   2474839dd9338a116fafd17c
```

Popular (but Broken) Crypto Hashes

- **MD5** – Message Digest 5
  - invented by Rivest
  - 128 bit output
  - Note: MD5 collisions easy to find
- **SHA-1** – Secure Hash Algorithm 1
  - U.S. government standard
  - inner workings similar to MD5
  - 160 bit output


ubuntu

Ubuntu-Core 13.04 (Raring Ringtail)

Select an image

For ARM hardware for which we do not ship preinstalled images, see ARM/Server/Install for detailed installation information.

A full list of available files can be found below.

```
Name                                       Last modified      Size  Description
Parent Directory                                                 -
MD5SUMS                                    25-Apr-2013 10:23  281
MD5SUMS.gpg                                25-Apr-2013 10:23  198
SHA1SUMS                                   25-Apr-2013 10:23  313
SHA1SUMS.gpg                               25-Apr-2013 10:23  198
SHA256SUMS                                 25-Apr-2013 10:23  409
SHA256SUMS.gpg                             25-Apr-2013 10:23  198
ubuntu-core-13.04-core-amd64.manifest      23-Apr-2013 17:30  2.9K
ubuntu-core-13.04-core-amd64.tar.gz        23-Apr-2013 17:30   38M
```

MD5SUMS text

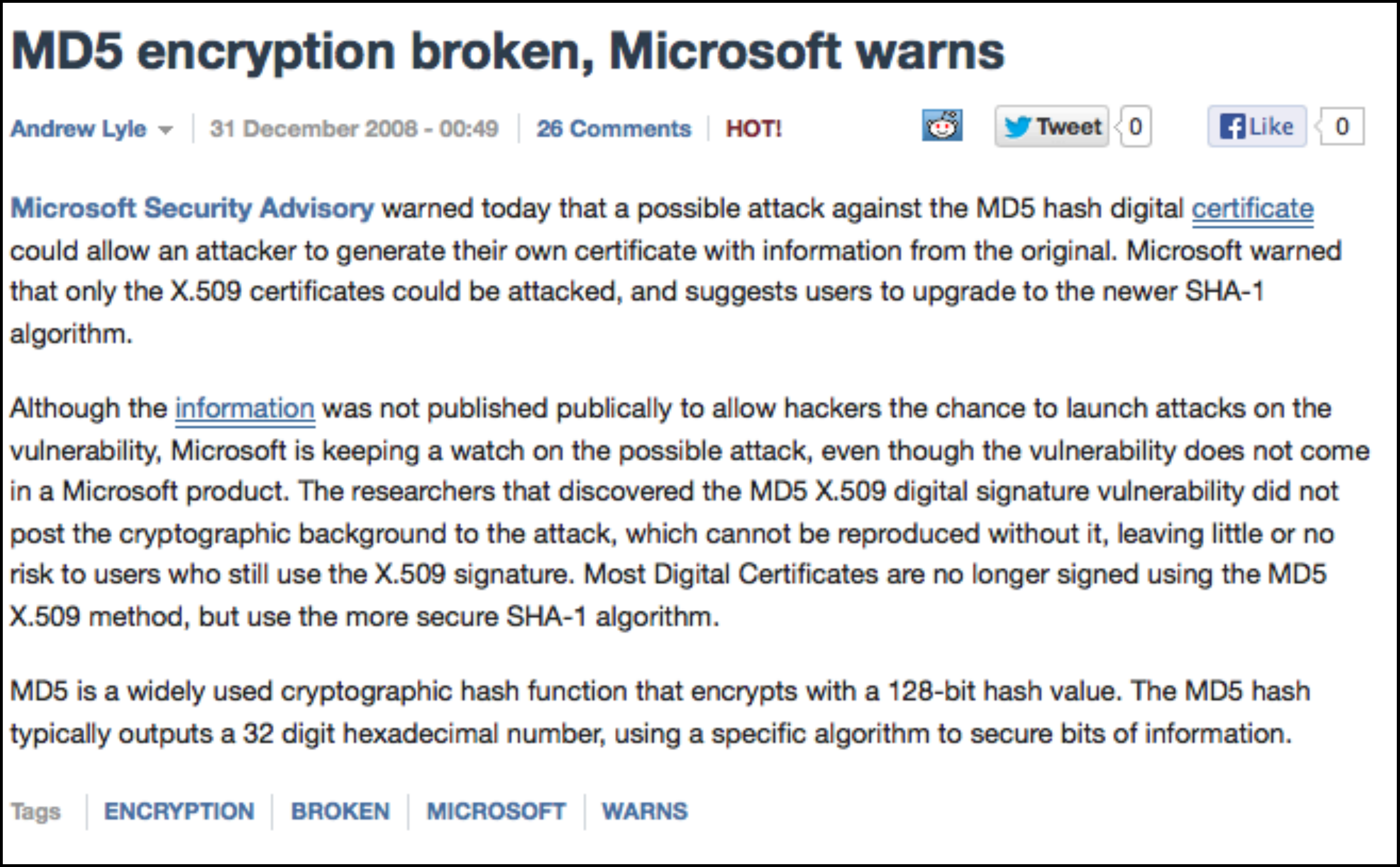
```
31125bf3134b4668ef5b0e93238cc922 *ubuntu-core-13.04-core-amd64.tar.gz
3480417a46bd9c53ca4594838fd9876e *ubuntu-core-13.04-core-armhf.tar.gz
f058338adedcad35e14e1443ef622740 *ubuntu-core-13.04-core-i386.tar.gz
c0b7a109824620122bfcc6062d4aeec3 *ubuntu-core-13.04-core-powerpc.tar.gz
```

`c0b7a109824620122bfcc6062d4aeec3`

MD5SUMS.gpg

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iEYEABECAAYFAlF5BCwACgkQRhgUM/u3VFHcjgCdGbqf2dS6VwTtiKeq0PHaOtAr
RnAAnj9kthXEVG7gjs9DCWpuHxJOZQyW
=XjUX
-----END PGP SIGNATURE-----
```



MD5 encryption broken, Microsoft warns

Andrew Lyle | 31 December 2008 - 00:49 | 26 Comments | HOT!

Tweet 0 | Like 0

Microsoft Security Advisory warned today that a possible attack against the MD5 hash digital certificate could allow an attacker to generate their own certificate with information from the original. Microsoft warned that only the X.509 certificates could be attacked, and suggests users to upgrade to the newer SHA-1 algorithm.

Although the information was not published publically to allow hackers the chance to launch attacks on the vulnerability, Microsoft is keeping a watch on the possible attack, even though the vulnerability does not come in a Microsoft product. The researchers that discovered the MD5 X.509 digital signature vulnerability did not post the cryptographic background to the attack, which cannot be reproduced without it, leaving little or no risk to users who still use the X.509 signature. Most Digital Certificates are no longer signed using the MD5 X.509 method, but use the more secure SHA-1 algorithm.

MD5 is a widely used cryptographic hash function that encrypts with a 128-bit hash value. The MD5 hash typically outputs a 32 digit hexadecimal number, using a specific algorithm to secure bits of information.

Tags | ENCRYPTION | BROKEN | MICROSOFT | WARNS

Broken Hashes, Broken Dreams

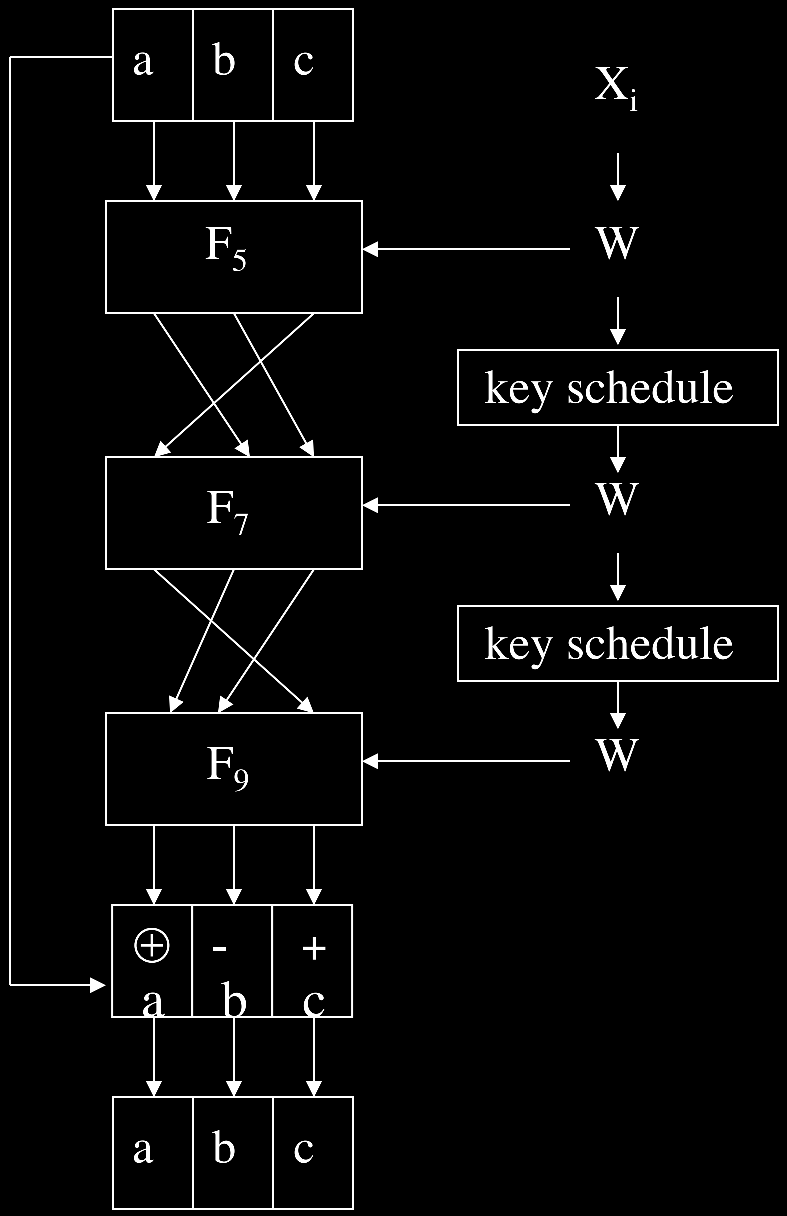
- MD5 collisions discovered
- Known shortcut attack for SHA-1.
- MD5 collision attack requires the attacker to control both hashed files.
  - Is this just a theoretical threat?
  - Is MD5 still safe for other uses?

Tiger Hash

- "Fast and strong"
- Designed by Ross Anderson and Eli Biham
- Design criteria
  - Secure
  - Optimized for **64-bit** processors
  - Easy replacement for MD5 or SHA-1

Tiger Hash

- Input divided into 512 bit blocks (padded)
  - similar to MD5/SHA-1
- Output is **192 bits** (three 64-bit words)
  - Truncate output if replacing MD5 or SHA-1
- Intermediate rounds are all 192 bits
- 4 S-boxes, each maps 8 bits to 64 bits
- A “key schedule” is used

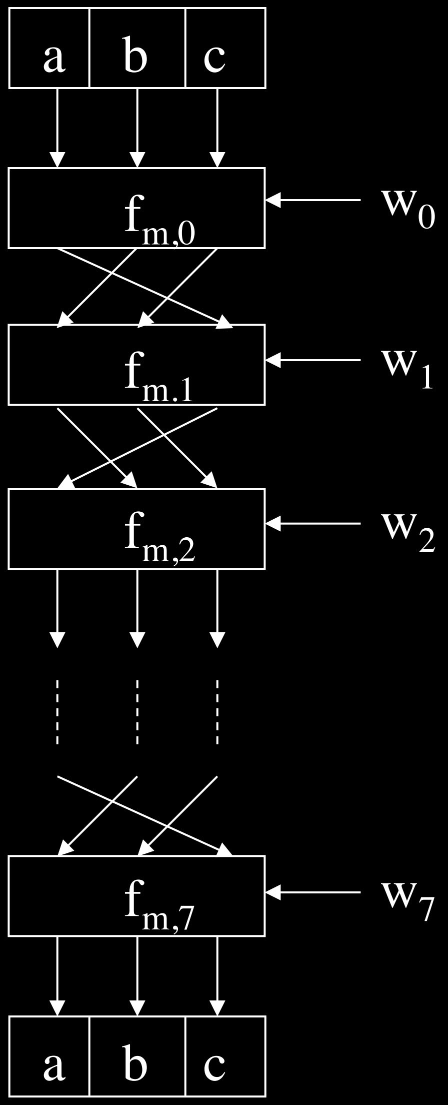


Tiger Outer Round

- Input is X
  - $X = (X_0, X_1, \ldots, X_{n-1})$
  - X is padded
  - Each X_i is 512 bits
- There are n iterations of diagram at left
  - One for each input block
- Initial (a,b,c) constants
- Final (a,b,c) is hash
- Looks like block cipher!

Tiger Inner Rounds

- Each F_m consists of precisely **8 rounds**
- 512 bit input W to F_m
  - $W=(w_0,w_1,\ldots,w_7)$
  - W is one of the input blocks X_i
- All lines are 64 bits
- The $f_{m,i}$ depend on the S-boxes (next slide)



Tiger Hash: One Round

- Each $f_{m,i}$ is a function of a,b,c,w_i and m
  - Input values of a,b,c from previous round
  - And w_i is 64-bit block of 512 bit W
  - Subscript m is multiplier
  - And $c = (c_0,c_1,\ldots,c_7)$
- Output of $f_{m,i}$ is
  - $c = c \oplus w_i$
  - $a = a - (S_0[c_0] \oplus S_1[c_2] \oplus S_2[c_4] \oplus S_3[c_6])$
  - $b = b + (S_3[c_1] \oplus S_2[c_3] \oplus S_1[c_5] \oplus S_0[c_7])$
  - $b = b * m$
- Each S_i is **S-box**: 8 bits mapped to 64 bits

Tiger Hash Key Schedule

- Input is X
  - $X=(x_0,x_1,\ldots,x_7)$
- Small change in X will produce large change in key schedule output

$x_0 = x_0 - (x_7 \oplus \text{0xA5A5A5A5A5A5A5A5})$

$x_1 = x_1 \oplus x_0$

$x_2 = x_2 + x_1$

$x_3 = x_3 - (x_2 \oplus ((\sim x_1) << 19))$

$x_4 = x_4 \oplus x_3$

$x_5 = x_5 + x_4$

$x_6 = x_6 - (x_5 \oplus ((\sim x_4) >> 23))$

$x_7 = x_7 \oplus x_6$

$x_0 = x_0 + x_7$

$x_1 = x_1 - (x_0 \oplus ((\sim x_7) << 19))$

$x_2 = x_2 \oplus x_1$

$x_3 = x_3 + x_2$

$x_4 = x_4 - (x_3 \oplus ((\sim x_2) >> 23))$

$x_5 = x_5 \oplus x_4$

$x_6 = x_6 + x_5$

$x_7 = x_7 - (x_6 \oplus \text{0x0123456789ABCDEF})$

Tiger Hash Summary (1)

- Hash and intermediate values are 192 bits
- 24 (inner) rounds
  - **S-boxes:** Claimed that each input bit affects a, b and c after 3 rounds
  - **Key schedule:** Small change in message affects many bits of intermediate hash values
  - **Multiply:** Designed to ensure that input to S-box in one round mixed into many S-boxes in next
- S-boxes, key schedule and multiply together designed to ensure strong **avalanche** effect

Tiger Hash Summary (2)

- Uses ideas from block ciphers
  - S-boxes
  - Multiple rounds
  - Mixed mode arithmetic
- At a higher level, Tiger employs
  - Confusion
  - Diffusion

HMAC

- Can compute a MAC of the message M with key K using a “hashed MAC” or **HMAC**
- HMAC is a **keyed hash**
  - Why would we need a key?
- How to compute HMAC?
- Two obvious choices: h(K,M) and h(M,K)
- Which is better?

HMAC

- Should we compute HMAC as h(K,M) ?
- Hashes computed in blocks
  - $h(B_1,B_2) = F(F(A,B_1),B_2)$ for some F and constant A
  - Then $h(B_1,B_2) = F(h(B_1),B_2)$
- Let M' = (M,X)
  - Then h(K,M') = F(h(K,M),X)
  - Attacker can compute HMAC of M' without K
- Is h(M,K) better?
  - Yes, but… if h(M') = h(M) then we might have h(M,K)=F(h(M),K)=F(h(M'),K)=h(M',K)

The Right Way to HMAC

- Described in RFC 2104
- Let B be the block length of hash, in bytes
  - B = 64 for MD5 and SHA-1 and Tiger
- ipad = 0x36 repeated B times
- opad = 0x5C repeated B times
- Then

$$\mathrm{HMAC}(M,K) = h(K \oplus \mathrm{opad}, h(K \oplus \mathrm{ipad}, M))$$

Hash Uses

- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- Anything you can do with symmetric crypto
- Also, many, many clever/surprising uses…

Online Bids

- Suppose Alice, Bob and Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
- They don’t trust that bids will stay secret
- A possible solution?
  - Alice, Bob, Charlie submit **hashes** h(A), h(B), h(C)
  - All hashes received and posted online
  - Then bids A, B, and C submitted and revealed
- Hashes don’t reveal bids (one way)
- Can’t change bid after hash sent (collision)
- But there is a flaw here…

Spam Reduction

- Spam reduction
- Before accept email, want proof that sender spent effort to create email
  - Here, effort == CPU cycles
- Goal is to limit the amount of email that can be sent
  - This approach will not eliminate spam
  - Instead, make spam more costly to send

Hashcash: Spam Reduction

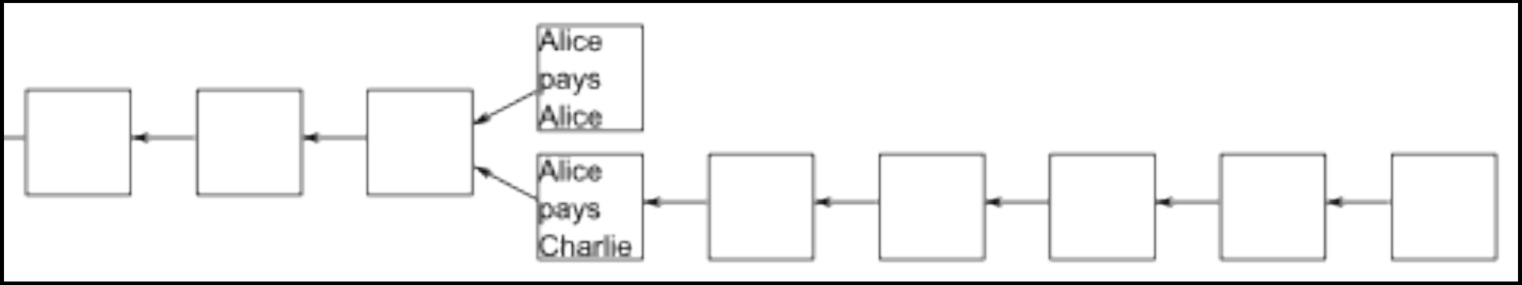
- Let M = email message
  R = value to be determined
  T = current time
- Sender must find **R** so that
  h(M,**R**,T) = (00…0,X), where
  N initial bits of hash value are **all zero**
- Sender then sends (M,**R**,T)
- Recipient accepts email, provided that…
  h(M,**R**,T) begins with N zeros

Hashcash: Work for Sender and Receiver

- Sender: h(M,R,T) begins with N zeros
- Recipient: verify that h(M,R,T) begins with N zeros
- **Work for sender:** about **2^N hashes**
- **Work for recipient:** always **1 hash**
- Sender's work increases exponentially in N
- Small work for recipient regardless of N
- Choose N so that…
  - Work acceptable for normal email users
  - Work is too high for spammers

Hashcash: Bitcoin Proof of Work

- The Bitcoin protocol uses proof of work to verify *block chains*.
  - determine transaction history
  - proof of work uses hashcash
- Prevents *double spending*, where Alice gives the same coin to Bob & Charlie.



MD5 collision lab

Download http://www.cs.sjsu.edu/~stamp/infosec/files/MD5_collision.zip. Open each file and read it with a postscript viewer.

Calculate the hash of each file.
Do they match?

Open up each file with a text editor.
Describe how this attack works.