COM307000 - Cryptography
Hash Functions

Dr. Anca Jurcut
E-mail: `anca.jurcut@ucd.ie`

School of Computer Science and Informatics
University College Dublin,
Ireland

Hash Function Motivation

- Suppose Alice signs M
  - Alice sends M and $S = \{M\}K_{APriv}$ to Bob
  - Bob verifies that $M = \{S\}K_{APub}$
  - Can Alice just send S?
- If M is big, $\{M\}K_{APriv}$ costly to ***compute*** & ***send***
- Suppose instead, Alice signs **h(M)**, where h(M) is a much smaller "fingerprint" of M
  - Alice sends M and $S = \{h(M)\}K_{APriv}$ to Bob
  - Bob verifies that $h(M) = \{S\}K_{APub}$

Crypto Hash Function

- 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 $\neq$ x such that h(y) = h(x)
  - **Strong collision resistance** — infeasible to find ***any*** x and y, with x $\neq$ y such that h(x) = h(y)
- Lots of collisions exist, but must be hard to find ***any***

Pre-Birthday Problem

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

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that any two (or more) have same birthday?
  - $1 - 365/365 \cdot 364/365 \cdot \cdot \cdot (365-N+1)/365$
  - 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, then 2^N different hash values are possible
- So, if you hash about $\mathrm{sqrt}(2^N) = 2^{N/2}$ values then you expect to find a collision
- **Implication?** "Exhaustive search" attack…
  - Secure N-bit hash requires $2^{N/2}$ work to "break"
  - Recall that secure N-bit symmetric cipher has work factor of 2^{N-1}
- Hash output length vs cipher key length?

Non-crypto Hash (1)

- Data $X = (X_1,X_2,X_3,\ldots,X_n)$, each X_i is a byte
- Define $h(X) = (X_1+X_2+X_3+\ldots+X_n) \bmod 256$
- 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

  $h(X) = (nX_1+(n-1)X_2+(n-2)X_3+\ldots+2\cdot X_{n-1}+X_n) \bmod 256$
- 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**
  - Such random errors unlikely to yield a collision
- But easy to ***construct*** collisions
  - In crypto, Trudy is the enemy, not “random”
- CRC has been mistakenly used where crypto integrity check is required (e.g., WEP)

Popular Crypto Hashes

- **MD5** — invented by Rivest (of course…)
  - 128 bit output
  - MD5 collisions easy to find, so it's broken
- **SHA-1** — a U.S. government standard, inner workings similar to MD5
  - 160 bit output
- Many other hashes, but MD5 and SHA-1 are the most widely used
- Hashes work by hashing message in blocks

Crypto Hash Design

- 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

Tiger Hash

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

Tiger Hash

- Like MD5/SHA-1, input divided into 512 bit blocks (padded)
- Unlike 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 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

Hash Function Motivation

- So, Alice signs **h(M)**
  - That is, Alice computes $S = \{h(M)\}K_{APriv}$
  - Alice then sends (M, S) to Bob
  - Bob verifies that $h(M) = \{S\}K_{APub}$
- What properties must h(M) satisfy?
  - Suppose Trudy finds M’ so that h(M) = h(M’)
  - Then Trudy can replace (M, S) with (M’, S)
- Does Bob detect this tampering?
  - No, since $h(M') = h(M) = \{S\}K_{APub}$

Tiger Hash Summary (2)

- Uses lots of 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)

Correct 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

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

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 serious flaw here…

Hashing for Spam Reduction

- Spam reduction
- Before accept email, want proof that sender had to “work” to create email
  - Here, “work” == 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

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), that is,
  initial N 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

Spam Reduction

- Sender: h(M,R,T) begins with N zeros
- Recipient: verify that h(M,R,T) begins with N zeros
- **Work for sender:** on average **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 amounts of email
  - Work is too high for spammers

Next…Secret Sharing