Hash Functions

Cryptographic 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)
- Many collisions exist, but cannot find any

Non-crypto Hash (1)

- Data $X = (X_0, X_1, X_2, \ldots, X_{n-1})$, each X_i is a byte
- Spse hash(X) = $X_0 + X_1 + X_2 + \ldots + X_{n-1}$
- Is this secure?
- Example: X = (10101010,00001111)
- Hash is 10111001
- But so is hash of Y = (00001111,10101010)
- 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
  - $h(X) = nX_0 + (n-1)X_1 + (n-2)X_2 + \ldots + 1 \cdot X_{n-1}$
- Is this hash secure? At least $h(10101010,00001111) \neq h(00001111,10101010)$
- But hash of (00000001,00001111) is same as hash of (00000000,00010001)
- Not secure, but it 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**
- Easy for Trudy to construct collisions
- CRC sometimes mistakenly used in crypto applications (WEP)

Popular Crypto Hashes

- **MD5** — invented by Rivest
  - 128 bit output
  - Note: MD5 collision recently found
- **SHA-1** — A US government standard (similar to MD5)
  - 160 bit output
- Many others hashes, but MD5 and SHA-1 most widely used
- Messages are hashed in blocks

Public Key Notation

- **Sign** message M with Alice's **private key**: $[M]_{Alice}$
- **Encrypt** message M with Alice's **public key**: $\{M\}_{Alice}$
- Then

$$\{[M]_{Alice}\}_{Alice} = M$$

$$[\{M\}_{Alice}]_{Alice} = M$$

Crypto Hash Motivation: Digital Signatures

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

Digital Signatures

- Digital signatures provide **integrity**
  - Like MAC and HMAC
- Why?
- Alice sends M and $S = [h(M)]_{Alice}$ to Bob
- If M changed to M′ or S changed to S′ (accident or intentional) Bob detects it:

  $h(M') \neq \{S\}_{Alice}$, $h(M) \neq \{S'\}_{Alice}$, $h(M') \neq \{S'\}_{Alice}$

Non-repudiation

- Digital signature also provides for **non-repudiation**
- Alice sends M and S = $[h(M)]_{Alice}$ to Bob
- Alice cannot "repudiate" signature
  - Alice cannot claim she did not sign M
- Why does this work?
- Is the same true of MAC?

Non-non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice computes **MAC** using symmetric key
- Stock drops, Alice claims she did not order
- Can Bob prove that Alice placed the order?
- **No!** Since Bob also knows symmetric key, he could have forged message
- **Problem:** Bob knows Alice placed the order, but he cannot prove it

Non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice **signs** order with her private key
- Stock drops, Alice claims she did not order
- Can Bob prove that Alice placed the order?
- **Yes!** Only someone with Alice's private key could have signed the order
- This assumes Alice's private key is not stolen (revocation problem)

Hashing and Signatures

- Alice signs h(M), sends M and $S = [h(M)]_{Alice}$ to Bob and Bob verifies $h(M) = \{S\}_{Alice}$
- Security depends on public key system **and** hash function
- Suppose Trudy can find collision: $M' \neq M$ with $h(M') = h(M)$
- Then Trudy can replace M with M′ and signature scheme is broken

Crypto Hash Function Design

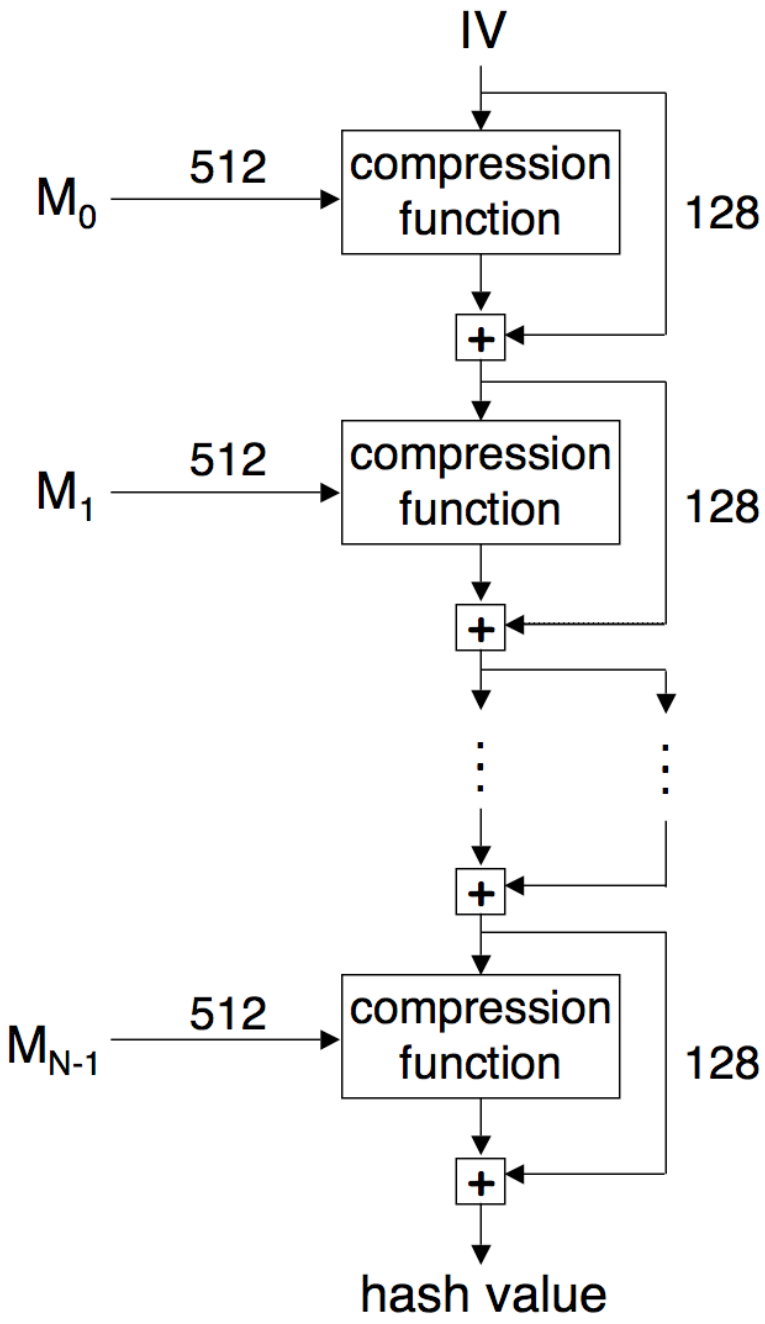
- Desired property: **avalanche effect**
  - Any change to input affects lots of output bits
- Crypto hash functions consist of some number of rounds
  - Analogous to block cipher in CBC mode
- Want security and speed
  - Avalanche effect after few rounds
  - But simple rounds

Crypto Hash Function Design

- Input data split into blocks
- **Compression function** applied to blocks
  - Current block and previous block output
  - Output for last block is the hash value
- For hashes we consider
  - Block size is 512 bits
  - Compression function output is 128 bits

Hash Function

- Input or "message" blocks $M_0, M_1, \ldots, M_{N-1}$
- Addition is mod 2^{32} per 32-bit word
- This is known as **Merkle-Damgard construction**



Crypto Hash: Fun Facts

- If msg is one 512-bit block: h(M) = f(IV,M) where f and IV known to Trudy
- For 2 blocks:

  $h(M) = f(f(IV,M_0),M_1) = f(h(M_0),M_1)$
- In general $h(M) = f(h(M_0,M_1,\ldots,M_{n-2}),M_{n-1})$
  - If h(M) = h(M′) then h(M,X) = h(M′,X) for any X
  - Implications for design of "hashed MAC"…

HMAC

- MAC: block cipher for integrity
- Can we use a hash function instead?
- A "hashed MAC", **HMAC**, of M with key K
  - Why is a key necessary?
- How to compute HMAC?
- Two obvious choices: h(K,M) and h(M,K)
- Which (if either) is better?

How to Compute HMAC?

- Should we compute HMAC as h(K,M) ?
- Hashes computed in blocks
- Recall $h(M_0,M_1) = F(h(M_0),M_1)$
- Let $M' = (M,X)$
  - Then $h(K,M') = F(h(K,M),X)$
  - Trudy can compute HMAC of M' without K
  - Defeats the purpose of HMAC

How to Compute HMAC?

- Should we compute HMAC as h(M,K) ?
  - Is this better than h(K,M) ?
- If h(M′) = h(M) then

  h(M,K) = F(h(M),K) = F(h(M′),K) = h(M′,K)
- In this case, Trudy can compute HMAC without knowing the key K
  - But collision must be known
  - Better than h(K,M), but we can do better

The Right Way to HMAC

- Described in RFC 2104
- Let B be the block length of hash, in bytes
- For popular hash functions, B = 64
  - SHA-1, MD5, Tiger, etc.
- Define

  ipad = 0x36 repeated B times

  opad = 0x5C repeated B times
- Then HMAC(M,K) = H(K⊕opad, H(K⊕ipad, M))

Hashing and Birthdays

- The "birthday problem" arises in many crypto contexts
- We discuss it in hashing context
  - And "birthday attack" on digital signature
- Then Nostradamus attack
  - Learn how to predict the future!
  - Works against any hash that uses Merkle-Damgard construction

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
  - Find N = 253

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that 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?
- No, it "should be" about sqrt(365) since compare **pairs** x and y

Of Hashes and Birthdays

- If h(x) is N bits, 2^N hash values
- Note that sqrt(2^N) = $2^{N/2}$
- So, hash $2^{N/2}$ inputs and find a collision
  - "Birthday attack" — an exhaustive search
- An N-bit symmetric cipher key requires at most 2^{N-1} work to "break"
- An N-bit hash: at most $2^{N/2}$ work to "break"

Signature Birthday Attack

- Suppose hash output is n bits
- Trudy selects evil message E
  - Wants to get Alice's signature on E
- Trudy creates innocent message I
  - Alice willing to sign message I
- How can Trudy use birthday problem?

Signature Birthday Attack

- Trudy creates $2^{n/2}$ variants of I
  - All have same meaning as I
  - Trudy hashes each: $h(I_0), h(I_1), \ldots$
- Trudy creates $2^{n/2}$ variants of E
  - All have same meaning as E
  - Trudy hashes each: $h(E_0), h(E_1), \ldots$
- By birthday problem, $h(I_j) = h(E_k)$, some j,k

Signature Birthday Attack

- Alice signs innocent message I_j
- Then Trudy has $[h(I_j)]_{Alice}$
- But $[h(I_j)]_{Alice} = [h(E_k)]_{Alice}$
- Alice unwittingly "signed" evil E_k
- Attack relies only on birthday problem

Online Bid Example

- Suppose Alice, Bob, Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
  - They do not trust that bids will be secret
  - Nobody willing to submit their bid
- Solution?
  - Alice, Bob, Charlie submit **hashes** h(A),h(B),h(C)
  - All hashes received and posted online
  - Then bids A, B and C revealed
- Hashes do not reveal bids (one way)
- Cannot change bid after hash sent (collision)

Online Bid

- This protocol is not secure!
- A forward search attack is possible
  - Bob computes h(A) for likely bids A
- How to prevent this?
- Alice computes h(A,R), R is random
  - Then Alice must reveal A and R
  - Trudy cannot try all A and R

Online Bid

- Spse B = $1000 and Bob submits h(B,R)
- When revealed, B = $1 and C = $2
- Bob wants to change his bid: B′ = $3
- Bob computes h(B′,R′) for different R′ until he finds h(B′,R′) = h(B,R)
  - How much work?
  - Apparently, about 2^n hashes required

Weak Collision Attack

- Hash sometimes used to commit
  - For example, online bid example
- Attack on weak collision resistance requires work of about 2^n hashes
- Collision attack is only about $2^{n/2}$
- Nostradamus attack solves weak collision problem with only about $2^{n/2}$ hashes
  - For some cases, such as online bid example
  - Applicable to any Merkle-Damgard hash

Trudy Predicts Future?

- Trudy claims she can predict future
- Jan 1, 2008, she publishes y, claiming y = h(x)
  - Where x has final S&P 500 index for 2008 and other predictions for 2009 and beyond
- Jan 1, 2009, Trudy reveals x, with y = h(x)
  - And x has S&P 500 index for Dec. 31, 2008 along with other rambling predictions for 2009
- Does this prove Trudy can predict future?

Trudy Predicts Future?

- Trudy specifies y in advance
- Let P be S&P 500 for Dec 31, 2008
- Assuming Trudy cannot predict future, she must find S so that y = h(P,S)
- Trudy can hash 2^n different S
  - But, we assume this is too much work
  - Is there any shortcut?

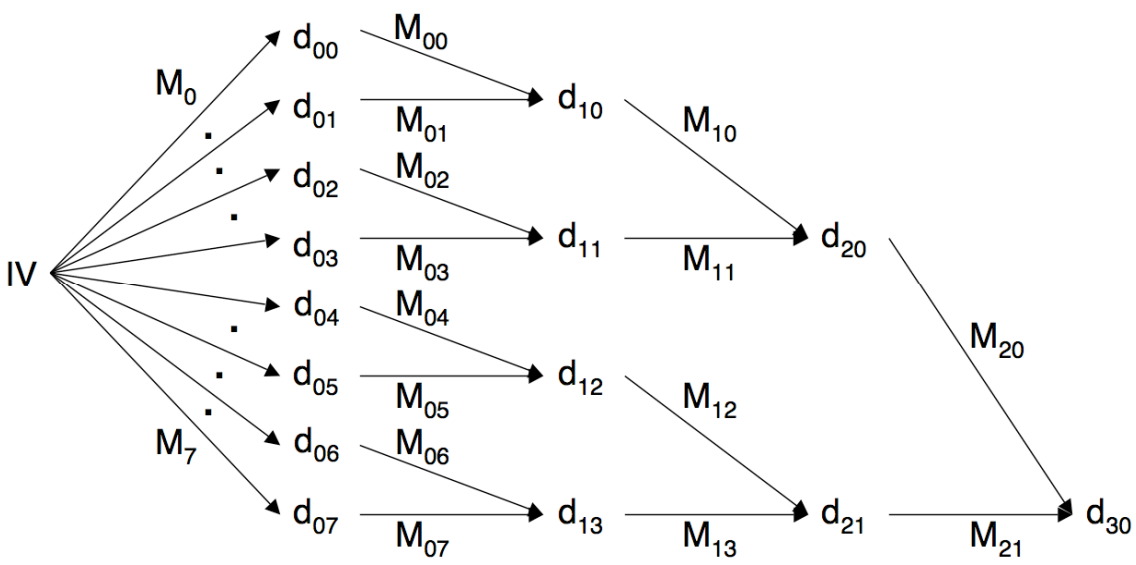
Nostradamus Attack

- Nostradamus (1503-1566) was a prophet
  - Some claim he predicted historical events
  - His predictive powers work best in retrospect
- Nostradamus attack
  - Trudy can predict the future
  - Convert 2^n pre-image problem into about $2^{n/2}$ collision attack (essentially)
  - Applies to any Merkle-Damgard hash function

Nostradamus Attack

- Computing collisions: each $2 \cdot 2^{n/2}$ work
  - Comparing one set to another set
- Pre-compute collisions in clever way
- This determines y, the hash value
- When we specify prefix P, we can "herd" collisions into hash value y
  - Suffix S determined in this process

Diamond Structure



- Choose M_0 randomly
- Compute $d_{00} = f(IV, M_0)$
- And $M_1, \ldots, M_7$

- Then find M_{00}, M_{01} that give collision:

  $d_{10} = f(d_{00}, M_{00}) = f(d_{01}, M_{01})$
- Continue: $y = d_{30}$ is pre-determined hash

Nostradamus Attack

- Pre-computation
  - Compute diamond structure of "height" 2^k
  - Choose $y = d_{k0}$ as hash of prediction
- When "prediction" is known, Trudy will
  - Let P be "prediction"
  - Select S′ at random, where (P,S′) one block
  - Until she finds $f(IV,P,S') = d_{0j}$ for some j

Nostradamus Attack

- Once such S' is found, Trudy has result
  - Follow directed path from d_{0j} to d_{k0}
- In previous diamond structure example, suppose Trudy finds $f(IV,P,S') = d_{02}$
- Then $h(P,S',M_{02},M_{11},M_{20}) = d_{30} = y$
  - Recall that y is hash of Trudy's "prediction"
- Let $x = (P,S',M_{02},M_{11},M_{20})$
- And x is Trudy's "prediction": P is S&P 500 index, S',M_{02},M_{11},M_{20} are future predictions

Nostradamus Attack

- How much work?
- Assuming diamond structure is of height 2^k and hash output is n bits
- Primary: $2 \cdot 2^{n/2}(2^k - 1) \approx 2^{n/2+k+1}$
  - Can reduce this to $2^{n/2+k/2+1}$
- Secondary: 2^{n-k}

Nostradamus Attack

- To minimize work, set primary work equal to secondary work, solve for k
- We have n/2 + k/2 + 1 = n – k which implies k = (n – 4)/3
- For MD4 or MD5, n = 128, so k = 41
- Diamond structure of height 2^{41}
- Total work is about 2^{87}

Nostradamus: Bottom Line

- Generic attack on any hash that uses Merkle-Damgard construction
- Not practical for 128-bit hash
  - Almost practical with small success prob
- Using hash to commit to something, is not quite as strong as it seems
- Next, MD4 followed by MD5
  - Must look at inner workings of these…