50.020 Security
Lecture 4: Applications for Hashing

Hash function applications (Introduction)

Applications for cryptographic hash

Hash functions provide interesting *one-way* function on data.

- Ideally, the hash value does not reveal any information about input

Can be used for:

- Commitment schemes
- Message integrity protection and authentication
- Storage of secrets (with some caveats)

Commitment schemes

Design challenge: Transparent bidding

- Assume n users are bidding on an item
  - Everyone can give one bid
  - Bids are compared once everyone gave his/her bid
  - Highest bid wins
- How can you collect the bids and determine the higher bid securely
  - without shared keys
- alternative game: rock, paper, scissors

Commitment schemes

- Solving both problems can be done using cryptographic hash $H(\cdot)$
- A two-phase protocol between Alice and Bob is used
- In the first phase:
  - Alice and Bob choose their action, e.g. m_a="rock", and computes $H(m_a)=c_a$
  - Alice and Bob exchange their commitments c
- In the second phase:
  - Both exchange their actual messages
  - Only the correct message will fit the commitment, so noone can cheat

Problems with this simple solution?

- The previous scheme has a simple problem
- There is a way how Alice and Bob can "invert" the commitment
- Any guesses?

Problems with this simple solution?

- The previous scheme has a simple problem
- There is a way how Alice and Bob can "invert" the commitment
- Any guesses?

- "Rock", "Paper", "Scissors" always hash to same values
- We need to make the input less predictable...

Cryptographic Padding

- So far, we assumed m was of correct length for block-based cryptographic functions (512 for SHA-1)
- If $l = |m|$ is too short, we have to apply *padding*
- Example: block size 10 characters, $m =$"Hello!"
- Can we just pad with random data? -> "Hello!X4Qa"

Cryptographic Padding

- So far, we assumed m was of correct length for block-based cryptographic functions (512 for SHA-1)
- If $l = |m|$ is too short, we have to apply *padding*
- Example: block size 10 characters, m ="Hello!"
- Can we just pad with random data? -> "Hello!X4Qa"

- Padding depends on function
  - for SHA-1, padding is a ONE followed by ZEROs and the length of the message
  - Other crypto functions require other schemes (more on that later)
  - Our commitment scheme would also require some randomness

Message authentication codes

Motivation MACs

- In last lecture, we had the "buy100" example
- Alice and Bob share a key k
- Using OTP or stream ciphers, they cannot guarantee integrity
- Using SHA directly also does not help
  - Attacker can compute new hash, flip bits as well
- Message authentication codes prevent this attack

Message authentication codes (MACs)

Requirements:

- Alice and Bob share k
- Alice wants to send m to Bob, can add some x
- Using x, Bob should verify integrity of m
- Both have access to cryptographic hash function H($\cdot$)
- How to construct from k,m, and H($\cdot$)?

Message authentication codes (MACs)

Requirements:

- Alice and Bob share k
- Alice wants to send m to Bob, can add some x
- Using x, Bob should verify integrity of m
- Both have access to cryptographic hash function H($\cdot$)
- How to construct from k,m, and H($\cdot$)?

- Secret as prefix: $x = H(k||m)$
- Secret as suffix: $x = H(m||k)$
- Alice will then send (m, x) to Bob

Which is better? $x = H(k||m)$ or $x = H(m||k)$?

- One of the two allows attacker to create valid MAC for a version of m with additional blocks at the end
  - i.e. attacker can produce valid $x' = H(k, m||m')$
- One of the two allows attacker to re-use MAC if second preimage can be found
  - i.e. attacker can reuse hash: $x = H(k, m')$, iff $H(m) = H(m')$
- Can you figure out which? (Assuming MD construction)

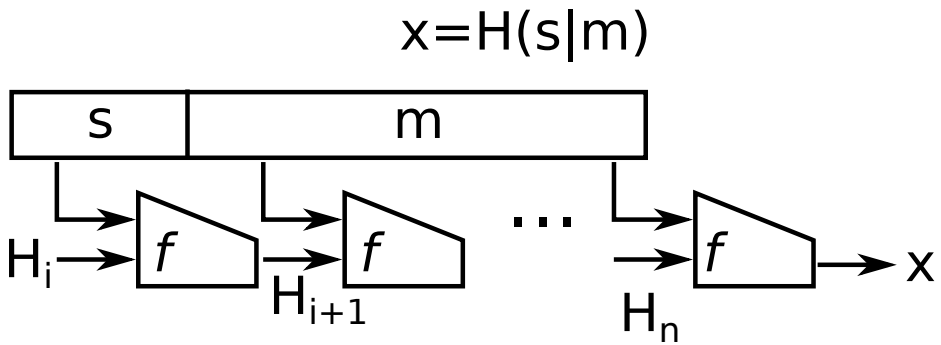
Which is better? $x = H(k||m)$ or $x = H(m||k)$?

- One of the two allows attacker to create valid MAC for a version of m with additional blocks at the end
  - i.e. attacker can produce valid $x' = H(k, m||m')$
- One of the two allows attacker to re-use MAC if second preimage can be found
  - i.e. attacker can reuse hash: $x = H(k, m')$, iff $H(m) = H(m')$
- Can you figure out which? (Assuming MD construction)

- Attack on prefix MAC: append another block to known MAC
  - $H(m||m') = H(m')$ with initial state $H_0 = H(m)$
- Attack on suffix MAC: find m' with $H(m') = H(m)$
  - Then, the original HMAC is also valid for m'

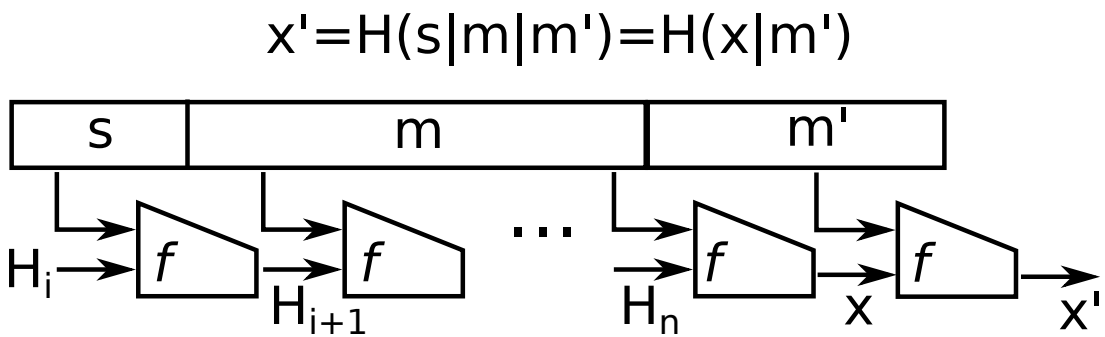
Details on attack on $x = H(s|m)$

- The attack exploits the fact that x captures the full internal state of the hash.
- Attacker can build on x to derive x'



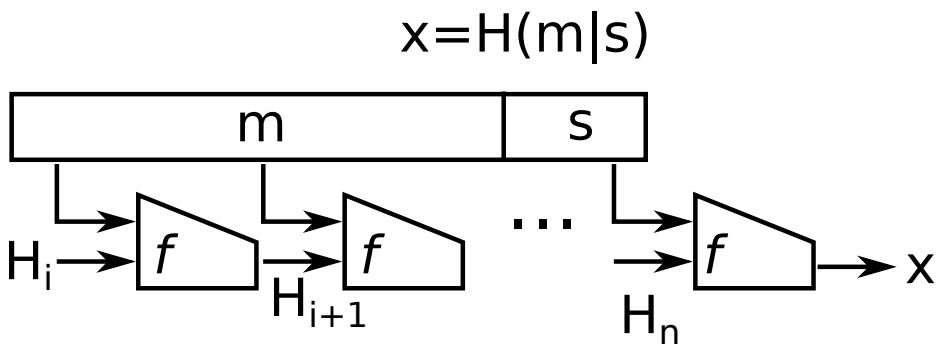
Details on attack on $x = H(s|m)$

- The attack exploits the fact that x captures the full internal state of the hash.
- Attacker can build on x to derive x'



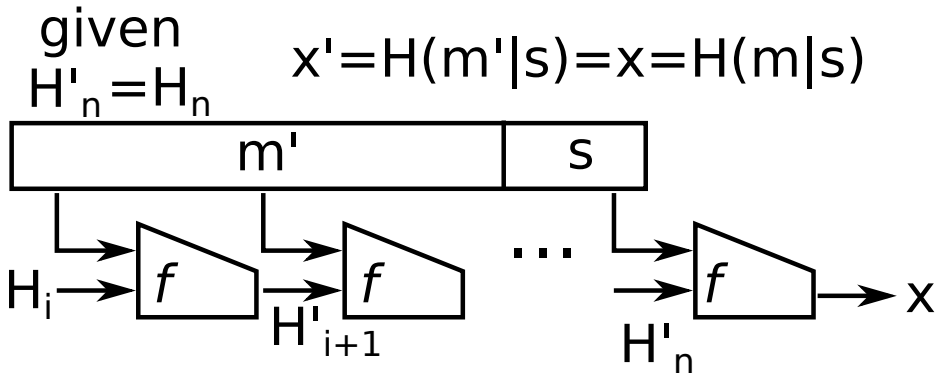
Details on attack on $x = H(m|s)$

- The attack exploits the fact x is valid for all m that hash to same value (second preimages)
- Attacker can simply re-use x



Details on attack on $x = H(m|s)$

- The attack exploits the fact x is valid for all m that hash to same value (second preimages)
- Attacker can simply re-use x



Hash-based MACs (HMACs)

- HMAC combines both prefix and suffix secrets to defeat attacks
- Construction: HMAC$(k, m) = H((k \oplus$ opad$) \,||\,$ H((k$\oplus$ipad) $||$ m))
  - k is a secret key padded with zeros
  - opad is outer padding (0x5c5c5c...5c5c)
  - ipad is inner padding (0x363636...3636)
- So, Alice sends $(m, HMAC(k, m))$ to Bob
  - Bob computes HMAC for m and k
  - Bob accepts message as authentic if HMAC is same as sent
  - Attacker cannot construct valid HMAC without k
  - Attacker cannot change m without changing HMAC

Storage of secrets

Storage of secrets

Consider the following problem

- You want to store a set of username and their passwords
  - `alice p4ssw0rd`
- Other users might be able to have read (or attackers copy data)
- How to protect the passwords of the users?

Storage of secrets

Consider the following problem

- You want to store a set of username and their passwords
  - `alice p4ssw0rd`
- Other users might be able to have read (or attackers copy data)
- How to protect the passwords of the users?

In our Linux installations, passwords are stored as SHA512 hashes in (/etc/shadow)

- When user inputs the password, it is hashed and compared with hash
  - `alice f1697e66a08b79532d5802a5cf6ffa4c`
- This is intended to keep the passwords secret
  - Can you think of ways to attack this scheme?

Finding Preimages

- Attacker has the hash values of passwords
  - Needs to find the original passwords
  - Sounds impossible?

Finding Preimages

- Attacker has the hash values of passwords
  - Needs to find the original passwords
  - Sounds impossible?

- Attacker creates a list of likely passwords
- Compares hash of each one with stolen hashes
- For short passwords, complete lists can be precomputed (Rainbow tables)
- Using rainbow tables, large sets of user/hash tuples can be processed quickly
- Example: Hashcat

Hashing other secrets

- Hashing is also used to anonymize other data (e.g., IDs, URL blacklists)
- Without salt (random number), *randomness* of the input might be too low again
- Example cases where this lead to unwanted results
  - Anonymization of URL black list entries
  - Anonymization of license plates in data reported by police [1]

[1] https://medium.com/@vijayp/of-taxis-and-rainbows-f6bc289679a1

Other examples for attacks on hashes

Yuval's square root attack (Collisions)

- Attacker wants Alice to sign a m text of his choice
- Attacker wants Alice to believe that she signed harmless text t
- Attacker generates n different variations m_i, t_j of m and t
- If there is one collision between m_i and t_j, attack is succesful
  - Alice checks t_j and signs the harmless text
  - The signature is also valid for m_i, as it has the same hash!

MD5 collisions Inc.

- Collision attack on MD5 to create an intermediate CA certificate
- Intermediate CA certicate needs to be signed by CA
- Can then be used to sign other certificates
- Harmless certificate was constructed that has same hash as intermediate CA
- Harmless certificate was then signed by victim CA
- Signature was attached to intermediate CA

Conclusions

- Cryptographic hash functions can be used for:
  - Commitment schemes
    - Salting or padding is required
  - Storage of secrets (carefully)
    - Salting is required
  - Construction of MACs, e.g. HMAC
    - k has to be integrated carefully