CS458: Introduction to Information Security

Notes 7: Hash Functions

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

October 11, 2018

Slides: Modified from Christof Paar and Jan Pelzl

Outline

- Why we need hash functions
- How does it work
- Security properties
- Algorithms
- Example: The Secure Hash Algorithm SHA-1
- SHA-3

Introduction to Hash Functions

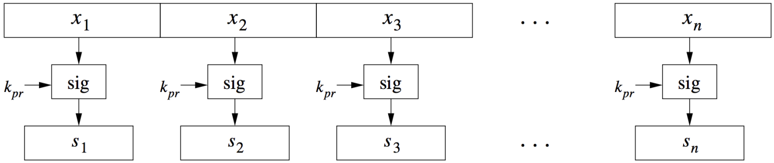
- Hash Functions
  - Auxiliary functions in cryptography
  - Used, e.g., for signatures, MACs, key derivation, RNGs, ...

Motivation: Signing Long Messages

- Suppose Bob signs x
- Bob sends x and $s = sig_{K_{pr,B}}(x)$ to Alice.
- Alice verifies that $ver_{K_{pub,B}}(x, s)$.
- Problem:
  - x is restricted in length, e.g., $|x| < 256$ Bytes
  - In practice the plaintext x will often be large.
- Q: How can we efficiently compute signatures of large messages?
- Divide the message x into blocks x_i of size less than the allowed input size of the signature algorithm, and sig each block separately

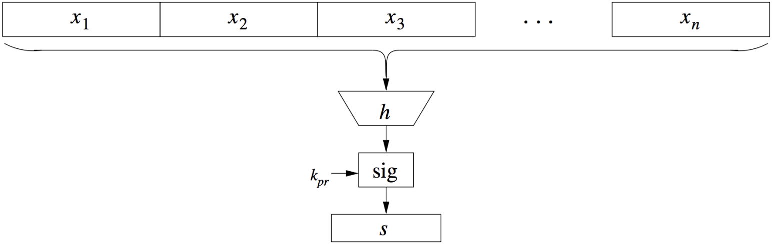
Motivation: Signing Long Messages

- **Problem**
  - Naïve signing of long messages generates a signature of same length.

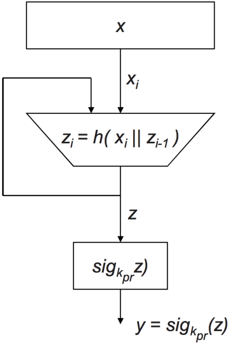


- Three Problems
  - Computational overhead
  - Message overhead
  - Security limitations
- **Solution**
  - Instead of signing the whole message, sig only a digest (=hash). Also secure, but much faster
  - i.e., somehow compress the message x prior to signing
- **Needed**
  - Hash Functions

Signing of long messages with a hash function



Signing of long messages with a hash function



- Notes:
  - x has fixed length
  - z, y have fixed length
  - z, x do not have equal length in general
  - $h(x)$ does not require a key.
  - $h(x)$ is public.

Basic Protocol for Digital Signatures with a Hash Function

Alice | | **Bob**

$$\xleftarrow{\quad k_{pub,B} \quad}$$

$$z = h(x)$$
$$s = \mathrm{sig}_{k_{pr,B}}(z)$$

$$\xleftarrow{\quad (x,s) \quad}$$

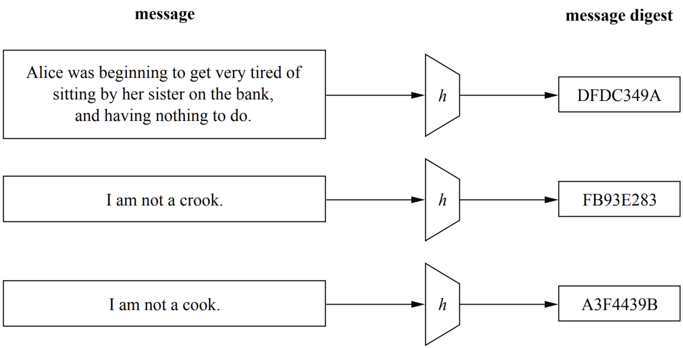
$$z' = h(x)$$
$$\mathrm{ver}_{k_{pub,B}}(s, z') = \text{true/false}$$

- z is a "fingerprint" of x or a "message digest"

Crypto Hash Function: Requirements

- Crypto hash function $h(x)$ must provide:
  1. Compression
     - Variable length into small, fixed length.
       i.e., Arbitrary message size $\Rightarrow$ Fixed output length
  2. Efficiency
     - $h(x)$ easy to compute for any x
  3. Preimage resistance (One-way)
     - For a given output z, it is impossible to find any input x such that $h(x)=z$, i.e., $h(x)$ is one-way
  4. Second preimage resistance (Weak collision resistance)
     - Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 s.t. $h(x_1)=h(x_2)$.
  5. Collision resistance (Strong collision resistance)
     - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.
- Actually, lots of collisions exist, but hard to find any

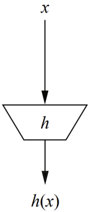
Principal input-output behavior of hash functions



- Able to apply a hash function to messages x of any size
- Output of a hash function is of fixed length and independent of the input length
- Computed fingerprint should be highly sensitive to all input bits. $\Rightarrow$ minor modifications to the input x, fingerprint should look very different

1^{st} Security properties of hash functions

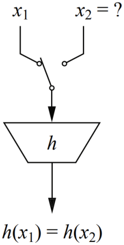
1. Preimage resistance (One-way)
    - For a given output z, it is impossible to find any input x such that $h(x)=z$, i.e., $h(x)$ is one-way
    - Bob sends $(e_K(x), sig_{K_{pr,B}}(z))$
        - Encrypts with AES and signs with RSA: $s = sig_{K_{pr,B}}(z) \equiv z^d \bmod n$
    - Eve uses Bob's public key to calculate $s^e \equiv z \bmod n$
    - If $h(x)$ is not one-way then $x = h^{-1}(z)$
        - Thus, the symmetric encryption of x is circumvented by the signature, which leaks the plaintext. $\Rightarrow h(x)$ should be a one-way function



preimage resistance

2^{nd} Security properties of hash functions

2. Second preimage resistance (Weak collision resistance)
   - Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 s.t. $h(x_1)=h(x_2)$.



second preimage
resistance

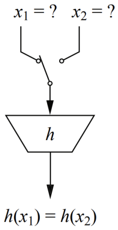
2^{nd} Collision Attack

- Assume Bob hashes and signs a message x_1.
- If Eve is capable of finding a second message x_2 such that $h(x_1) = h(x_2)$, she can run the following substitution attack

Alice		**Eve**		**Bob**
			$\xleftarrow{k_{pub,B}}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x_2, s)}$	$\lightning$ substitute	$\xleftarrow{(x_1, s)}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s, z) = \mathrm{true}$				

- There is always x_2 such that $h(x_1) = h(x_2)$ but it should be difficult to find
- "weak" collision, requires exhaustive search

3. Collision resistance (Strong collision resistance)
   - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.



collision resistance

Collision Attack

- Eve starts with two messages:

  x_1= `Transfer $10 into Eve's account`

  x_2 = `Transfer $10,000 into Eve's account`
- She alters x_1 and x_2 at "nonvisible" locations and continues until the condition $h(x_1) = h(x_2)$ is fulfilled.
- With the two messages, she can launch the following attack

Alice		**Eve**		**Bob**
			$\xleftarrow{k_{pub,B}}$	
			$\xrightarrow{x_1}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x_2,s)}$	$\lightning$ substitute	$\xleftarrow{(x_1,s)}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s,z) = \text{true}$				

Collision Attacks and the Birthday Paradox

- it turns out that collision resistance causes most problems and more difficult
- Collision attacks are much harder to prevent than 2^{nd} preimage attack.
- Q: Can we have hash function without collisions?
- Since $|X| >> |Z| \Rightarrow$ collision must exist. ("Pigeonhole Principle")

  $\Rightarrow$ We must make collision very hard to find!

2^{nd} preimage attack with brute-force

- If $|Z| = 2^{80}$, where $|h(x)| = n = 80$

  $\Rightarrow$ attack requires $\approx 2^{80}$ steps until to find a collision

Collision Attack

- It turns out that collision resistance causes most problems and more difficult to achieve
  - How hard is it to find a collision with a probability of 0.5?
  - Related Problem: How many people are needed such that two of them have the same birthday with a probability of 0.5?
    - $P(\text{no collision among 2 people}) = 1-\frac{1}{365}$
    - $P(\text{no collision among 3 people}) = (1-\frac{1}{365})(1-\frac{2}{365})$
    - $\ldots$
    - $P(\text{no collision among } t \text{ people}) = \prod_{i=1}^{t-1} (1-\frac{i}{365})$
    - for $t = 23$, $\prod_{i=1}^{22} (1-\frac{i}{365}) = 0.507 \approx 50\%$
  - No! Not $\frac{365}{2} = 183$. 23 are enough! This is called the **birthday paradox** (Search takes $\approx \sqrt{2^n}$ steps for 50% probability collision)
  - To deal with this paradox, hash functions need a output size of at least 160 bits

Uses of Hash Functions

- Authentication (HMAC) and Message integrity (HMAC)
  - Hash-based message authentication code
  - Keyed hash $h(k,x)$
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- "Proof of work"
- Securing passwords
- Digital certificates
- Building block of many protocols

Popular Crypto Hashes

- Many other hashes, but MD5 and SHA-1 are the most widely used.
- MD5: message-digest algorithm
  - *128* bit output
  - MD5 collisions easy to find, so it's broken.

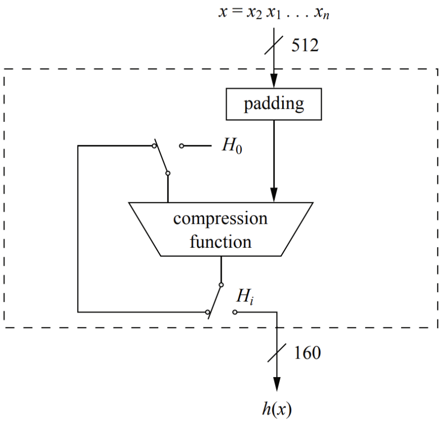
Popular Crypto Hashes

- SHA-1: Secure Hash Algorithm 1 - designed by NSA, inner workings similar to MD5.
  - 160 bit output
  - SHA-1 is no longer considered secure against well-funded opponents[1]
  - NIST issued revised FIPS 180-2 in 2002
    - Adds 3 additional versions of SHA
    - SHA-256, SHA-384, SHA-512
    - With 256/384/512-bit hash values
    - Same basic structure as SHA-1 but greater security
    - these hash algorithms are known as SHA-2
  - The most recent version is FIPS 180-4 (August 2015) which added two variants of SHA-512 with 224-bit and 256-bit hash sizes
  - Recommended to replace by SHA-2 or SHA-3

[1] Announcing the first SHA1 collision

SHA-1 High Level Diagram

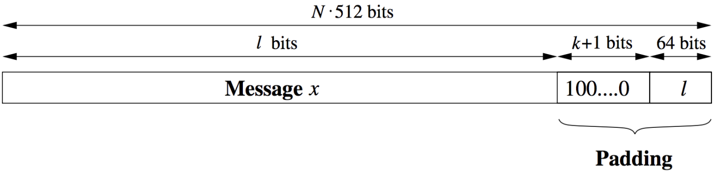
- Compression function consists of *80* rounds which are divided into four stages of *20* rounds each



- the initial value H_0 is set to a predefined constant.

SHA-1: Padding

- Message x has to be padded to fit a size of a multiple of 512 bit.
  - Let x with a length of l bit
  - To obtain an overall message size of a multiple of 512 bits
    - Append a single 1 followed by k zero bits and the binary 64-bit representation of l.
    - Consequently, the number of required zeros k is given by $k \equiv 512 - 64 - 1 - l = 448 - (l + 1) \bmod 512$



SHA-1: Padding: Example

- Given is the message *abc* consisting of three 8-bit ASCII characters with a total length of $l = 24$ bits:

$$\underbrace{\texttt{01100001}}_{\texttt{a}} \quad \underbrace{\texttt{01100010}}_{\texttt{b}} \quad \underbrace{\texttt{01100011}}_{\texttt{c}}.$$

We append a "1" followed by $k = 423$ zero bits, where k is determined by

$$k \equiv 448 - (l+1) = 448 - 25 = 423 \bmod 512.$$

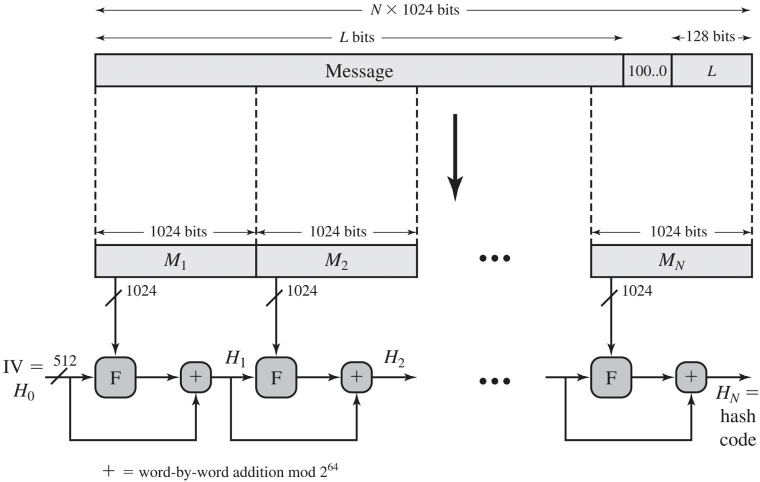
Finally, we append the 64-bit value which contains the binary representation of the length $l = 24_{10} = 11000_2$. The padded message is then given by

$$\underbrace{\texttt{01100001}}_{\texttt{a}} \quad \underbrace{\texttt{01100010}}_{\texttt{b}} \quad \underbrace{\texttt{01100011}}_{\texttt{c}} \quad 1 \quad \underbrace{00...0}_{423 \text{ zeros}} \quad \underbrace{00...011000}_{l=24}.$$

Message Digest Generation Using SHA-512

- **Step 1: Append padding bits**: message length is congruent to *896* $modulo$ *1024*
- **Step 2: Append length**: as a block of *128* bits being an unsigned *128*-bit integer length of the original message (before padding).
- **Step 3: Initialize hash buffer**: *512*-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight *64*-bit registers
- **Step 4: Process the message in *1024-bit* (*128-word*) blocks**: The heart of the algorithm is a module that consists of *80* rounds; this module is labeled F in Figure in next slide.
- **Step 5: Output**: After all N *1024-bit* blocks have been processed, the output from the N^{th} stage is the *512-bit* message digest.

Message Digest Generation Using SHA-512



SHA-3

- SHA-2 shares same structure and mathematical operations as its predecessors and causes concern
- Due to time required to replace SHA-2 should it become vulnerable, NIST announced in 2007 a competition to produce SHA-3
- SHA-3 Requirements:
  - Must support hash value lengths of 224, 256, 384, and 512 bits
  - Algorithm must process small blocks at a time instead of requiring the entire message to be buffered in memory before processing it
- SHA-3 standard was released by NIST on August 5, 2015

Lessons Learned

- Hash functions are keyless. The two most important applications are: digital signatures and in message authentication codes such as HMAC
- The 3 security requirements for hash functions are one-wayness, second preimage resistance and collision resistance
- Hash functions should have at least 160-bit output length in order to withstand collision attacks; 256 bit or more is desirable for long-term security
- Some security weaknesses have been found in SHA-1, and it is being phased out. The SHA-2 algorithms appear to be more secure but also start to be questionable
- The SHA-3 competition resulted in new standardized hash functions