Hash Functions and MACs

Cryptography

School of Engineering and Technology
CQUniversity Australia

Prepared by Steven Gordon on 15 Apr 2020,
hash.tex, r1851

Summary of Authentication Primitives

- Two types of general hash functions:
- Unkeyed hash function, $h = \mathrm{H}(M)$
  - Also simply called (cryptographic) hash function
  - Output hash value, h, also called *message digest*, *digital fingerprint*, or *imprint*
  - Primary function: MDC
- Keyed hash function, $h = \mathrm{H}(K, M)$
  - Output h often called *code*, *tag* or *MAC*
  - Primary function: MAC1

We will mainly refer to each technique as a *hash function* or *Message Authentication Code*. Chapter 9 of the Handbook of Applied Cryptography explains the different classifications of hash functions.

Contents

Hash Functions for Cryptography

- Hash function or algorithm $\mathrm{H}()$:
  - Input: variable-length block of data M
  - Output: fixed-length, small, hash value, h, where $h = \mathrm{H}(M)$
  - Another name for hash value is digest
  - Output hash values should be evenly distributed and appear random
- A secure, cryptographic hash function is practically impossible to:
  - Find the original input given the hash value
  - Find two inputs that produce the same hash value

A hash function is an algorithm that usually takes any sized input, like a file or a message, and produces a short (e.g. 128 bit, 512 bit) random looking output, the hash value. If you apply the hash function on the same input, you will always get the exact same hash value as output. In practice, if you apply the hash function on two different inputs, you will get two different hash values as output.

Applications of Hash Functions

- Message authentication
- Digital signatures
- Storing passwords
- Signatures of data for malicious behaviour detection (e.g. virus, intrusion)
- Generating pseudorandom number

Hash functions are important in many areas of security. They are typically used to create a fingerprint/signature/digest of some input data, and then later that fingerprint is used to identify if the data has been changed. However they also have uses for hiding original data (storing passwords) and generating random data. Different applications may have slightly different requirements regarding the security (and performance) properties of hash functions.

Design Approaches for Hash Functions

Based on Block Ciphers Well-known and studied block ciphers are used with a mode of operation to produce a hash function. Generally, less efficient than customised hash functions.

Based on Modular Arithmetic Similar motivation as to basing on block ciphers, but based on public key principles. Output length can be any value. Precautions are needed to prevent attacks that exploit mathematical structure.

Customised Hash Functions Functions designed for the specific purpose of hashing. Disadvantage is they haven't been studied as much as block ciphers, so harder to design secure functions.

Designing hash functions based on existing cryptographic primitives is advantageous in that existing knowledge and implementations can be re-used. However as more time has been spent studying customised hash functions, they are now the approach of choice due to their security and efficiency.

Selected Cryptographic Hash Functions

Primitive	Output Length	Classification	
		Legacy	Future
SHA-2	256, 384, 512, 512/256	✓	✓
SHA-3	256, 384, 512	✓	✓
SHA-3	SHAKE128, SHAKE256	✓	✓
Whirlpool	512	✓	✓
BLAKE	256, 384, 512	✓	✓
RIPEMD-160	160	✓	✗
SHA-2	224, 512/224	✓	✗
SHA-3	224	✓	✗
MD5	128	✗	✗
RIPEMD-128	128	✗	✗
SHA-1	160	✗	✗

Credit: ECRYPT CSA Algorithms, Key Size and Protocols Report, 2018

The figure on slide 7 shows selected hash functions, classified for legacy or future use. It is taken from the ECRYPT-CSA 2018 report on Algorithms, Key Sizes and Protocols. The authors classified hash functions as legacy, meaning secure for near future, and future, meaning secure for medium term. It includes history hash functions no longer recommended, such as MD5, RIPEMD-128 and SHA-1. There are many other hash functions. Wikipedia has a nice comparison.

Contents

Pre-image of a Hash Value (definition)

For hash value $h = \mathrm{H}(x)$, x is pre-image of h. As H is a many-to-one mapping, h has multiple pre-images. If H takes a b-bit input, and produces a n-bit hash value where $b > n$, then each hash value has 2^{b-n} pre-images.

A hash function takes a single input and produces a single output. The output is the hash value and the input is the pre-image of that hash value.

Hash Collision (definition)

A collision occurs if $x \neq y$ and $\mathrm{H}(x) = \mathrm{H}(y)$. Collisions are undesirable in cryptographic hash functions.

We will show shortly that collisions should be practically impossible to be found by an attacker.

Number of Collisions (exercise)

If H_1 takes fixed length 200-bit messages as input, and produces a 80-bit hash value as output, are collisions possible?

Requirements of Cryptographic Hash Functions

Variable input size: H can be applied to input block of any size

Fixed output size: H produces fixed length output

Efficiency: $\mathrm{H}(x)$ relatively easy to compute (practical implementations)

Pseudo-randomness: Output of H meets standard tests for pseudo-randomness

Properties: Satisfies one or more of the properties: Pre-image Resistant, Second Pre-image Resistant, Collision Resistant

Pre-image Resistant Property (definition)

For any given h, it is computationally infeasible to find y such that $\mathrm{H}(y) = h$. Also called the *one-way property*.

Informally, it is hard to inverse the hash function. That is, given the output hash value, find the original input message.

Second Pre-image Resistant Property (definition)

For any given x, it is computationally infeasible to find $y \neq x$ with $\mathrm{H}(y) = \mathrm{H}(x)$. Also called *weak collision resistant* property.

To break this property, the attacker is trying to find a collision. That is, two input messages x and y that produce the same output hash value. Importantly, the attacker cannot choose x. They are given x and must find a different message y that produces a collision.

Collision Resistant Property (definition)

It is computationally infeasible to find any pair (x, y) such that $\mathrm{H}(x) = \mathrm{H}(y)$. Also called *strong collision resistant* property.

To break this property, again the attacker is trying to find a collision. However in this case the attacker has the freedom to find *any* messages x and y that produce a collision. This freedom makes it easier for the attacker to perform an attack against this property than against the Second Pre-image Resistant property.

Required Hash Function Properties for Different Applications

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

* Resistance required if attacker is able to mount a chosen message attack

Credit: Table 11.2 . In W. Stallings, "Cryptography and Network Security: Principles and Practice", 7th Edition, Pearson Education, 2017.

The figure on slide 16 (Table 11.2 from Stalling's Cryptography and Network Security, 7th Ed) shows which of the three properties are required to meet the security requirements of different applications. For example, if a hash function is to be used for the purpose of a digital signature, then the Preimage Resistant and Second Preimage Resistant properties are required, and the Collision Resistant property is also required if the attack has the ability to chosen a message and have it signed (hashed) by a user.

These properties and the applications will be investigated later when looking at authentication.

Brute Force Attacks on Properties

- Pre-image and Second Pre-image Attack
  - Find a y that gives specific h; try all possible values of y
  - With b-bit hash code, effort required proportional to 2^b
- Collision Resistant Attack
  - Find any two messages that have same hash values
  - Effort required is proportional to $2^{b/2}$
  - Due to birthday paradox, easier than pre-image attacks

Brute Force Attack on Hash Function (exercise)

Consider a hash function to be selected for use for digital signatures. Assume an attacker has compute capabilities to calculate 10^{12} hashes per second and is prepared to wait for approximately 10 days for a brute attack. Find the minimum hash value length that the hash function should support, such that a brute force is not possible.

Contents

Unkeyed and Keyed Hash Functions

- Hash functions have no secret key
  - Can be referred to as unkeyed hash function
  - Also called Modification Detection Code
- A variation is to allow a secret key as input, in addition to the message
  - $h = \mathrm{H}(K, M)$
  - Keyed hash function or Message Authentication Code (MAC)
- Hashes and MACs can be used for message authentication, but hashes also used for multiple other purposes
- MACs are more common for authentication messages

Design Approaches for MACs

Based on Block Ciphers CBC-MAC, OMAC, PMAC,

Customised MACs MAA, MD5-MAC, UMAC, Poly1305

Based on Hash Functions HMAC

The motivation for different design approaches is similar to that for hash function design approaches.

Computation Resistance of MAC (definition)

Given one or more text-tag pairs, $[x_i, \text{MAC}(K, x_i)]$, computationally infeasible to compute any text-tag pair $[y, \text{MAC}(K, y)]$, for a new input $y \neq x_i$

Assume an attacker has intercepted messages (text) and the corresponding MACs (tags). They have i such text-tag pairs. Now there is a new message y. It should be practically impossible for the attacker to find the corresponding tag of y, that is, $\text{MAC}(K, y)$.

Security of MACs

- Brute Force Attack on Key

  Attacker knows $[x_1, T_1]$ where $T_1 = MAC(K, x_1)$Key size of k bits: brute force on key, 2^kBut . . . many tags match T_1For keys that produce tag T_1, try again with $[x_2, T_2]$Effort to find K is approximately 2^k

- Brute Force Attack on MAC value

  For x_m, find T_m without knowing KSimilar effort required as one-way/weak collision resistant property for hash functionsFor n bit MAC value length, effort is 2^n

- Effort to break MAC: $\min(2^k, 2^n)$