

HASH FUNCTIONS & BASIC CIPHERS

Luke Anderson

luke@lukeanderson.com.au

10th March 2016

University Of Sydney



Overview

1. Crypto-Bulletin

2. Introduction & Context

2.1 Applied Cryptography

2.2 One Way Functions

Functions

One Way Functions

3. Hash Functions

3.1 Introduction

Properties

Applications

Examples

3.2 Attacks

3.3 Construction

4. Message Authentication Codes (MACs)

CRYPTO-BULLETIN

'Creepy' Aussie Anon sentenced for hacking websites

<http://www.itnews.com.au/news/creepy-aussie-anon-sentenced-for-hacking-websites-416485>

Why you should care about Australia's new defence trade controls

<http://www.itnews.com.au/feature/why-you-should-care-about-australias-new-defence-trade-controls-416446>

UNSW team wins Cyber Security Challenge

<http://www.itnews.com.au/news/unsw-team-wins-cyber-security-challenge-415599>

INTRODUCTION & CONTEXT

Context: Applied Cryptography

Cryptography is the study of mathematical techniques related to the design of ciphers.

Cryptanalysis is the study of breaking them.

Cryptology (or crypto) is the study of both.

Crypto building blocks are otherwise known as cryptographic primitives.
for example:

- *hash functions*
- block ciphers
- stream ciphers
- digital signatures
- random number generators

Functions

A function $f : X \rightarrow Y$ is defined by:

- The **domain**, a set $X = \{x_1, x_2, \dots, x_n\}$.
- The **codomain**, a set $Y = \{y_1, y_2, \dots, y_m\}$.
- A rule f assigning each element of X to an element of Y .

When $f : X \rightarrow Y$ is a function:

- The **image** of $x \in X$ is called $f(x)$, an element of Y .
- The **range** of f is the set of all images, and is a subset of Y .
- If $f(x) = y$, then x is called a **preimage** of y .
- The set of all preimages of y is written $f^{-1}(\{y\})$.

Functions: Example

Example: let $f : \{-1, 0, 1\} \rightarrow \{0, 1, 2\}$ be defined by $f(x) = x^2$.

- $f(-1) = 1$, $f(0) = 0$, and $f(1) = 1$.
- The preimage of 1 is $f^{-1}(\{1\}) = \{-1, 1\}$.
- The preimage of 2 is $f^{-1}(\{2\}) = \{\}$.
- The range of f is $\{0, 1\}$.

One Way Functions

Write $\{0, 1\}^n$ for the set of all binary strings of length n . For example:

- $\{0, 1\}^1 = \{0, 1\}$.
- $\{0, 1\}^2 = \{00, 01, 10, 11\}$.

A function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is one way (a OWF) if:

- It is “easy” to compute $f(x)$ for all $x \in X$
- It is “computationally infeasible” to find a preimage.

Intuitively:

- Given x , it is easy to compute $f(x)$
- Given $f(x)$, it is hard to compute x

One Way Function Examples

EXAMPLE

1. Write a message on the side of a plate: x
2. Smash the plate: $f(x)$
3. Finding the inverse $f^{-1}(x)$ is difficult (but not impossible)

Data Encryption Standard Cipher

1. $f(x) = \text{DES}(m, k) = c$
2. Given c and m , it is difficult to find k

HASH FUNCTIONS

Hash Functions

A hash function, h , is an efficiently computable mapping of arbitrarily long strings to short fixed length strings.

Minimum properties:

- Compression
Typically any number of bits to < 512 bits
e.g. MD5, SHA256, SHA512
- Ease of computation
Given h and x , $h(x)$ is easy to compute.

Keyed Hash Functions

Some hash functions take both a key (k) and a message (m)

$$\text{MAC}_k(m) = h(m, k)$$

They are also called message authentication codes (**MAC**) or hash-based message authentication codes (**HMAC**).

Properties of Secure Hash Functions

Not all hash functions are secure. In cryptography we consider *secure* hash functions.

Let $h : X \rightarrow Y$ be a hash function. In order to be *secure*, it must satisfy the following properties:

#1. Preimage Resistance

Given y it is “hard” to find a preimage x such that $h(x) = y$.

#2. Second Preimage Resistance

Given a particular x (and hence y), it is “hard” to find $x' \neq x$ such that: $h(x') = h(x) = y$.

#3. Collision Resistance

It is “hard” to find any pair $x \neq x'$ such that $h(x) = h(x')$.

Note: $3 \Rightarrow 2$ since $!2 \Rightarrow !3$

Properties of Secure Hash Functions

A **one way hash function** satisfies #1 and #2

A **collision resistant hash functions** satisfies #3 (and hence #2)

VERY USEFUL!

Hash functions are extremely useful for confirmation of knowledge without revealing what you know.

Rather than sending Alice a secret across the Internet, just send the hash of it. If Alice knows the secret, she can hash it and verify that you know it too.

This is much safer and efficient than sending the secret, which can of course be **intercepted** (provided the hash function is strong).

Hash Function Applications

Password Files

e.g. `/etc/shadow` on UNIX

Instead of storing the password in cleartext, store the hash. Then just compute the hash whenever someone tries to log in and compare to what was stored.

If the password file gets stolen, the hash needs to be reversed before the attacker can use the passwords (“cracking passwords”).

VIRUS PROTECTION & HOST INTRUSION DETECTION

e.g. Tripwire

- For each file x , $h(x)$ is stored off system.
- Periodically hash all files and check the hashes match
- Property #2 is critical as it should be hard to find x' such that $h(x) = h(x')$ (otherwise viruses could hide)

Real World Hash Functions

| Name | bits | h("lolcats") |
|--------|------|--|
| MD2 | 128 | 4301aae7e3e791826b53b952859d0a14 |
| MD4 | 128 | 52bb2839f24583f5af2fe74522db3e2e |
| MD5 | 128 | c8ba0a4b74948d105bdb6f77b77a432e |
| RIPEMD | 160 | 3dd2dec7cecec77219f644788e81ff26d328423c |
| SHA-1 | 160 | ec9c175f8e3780cec9e93b66aea4f98b200764de |
| SHA224 | 224 | c995ce647c889fdc3bbc7c8e4b43b3f5b5c3faf1525b640abc60ce54 |
| SHA256 | 256 | e06297effe5bcf6af177cead11f5c5d4a73777590a7a98a464287d5a6a7cdc2a 14c41e98d0fb9b357922274adb9f70352f601d7b56aac8e4... |
| SHA384 | 384 | ...39ed860b634b31a7c0f56e3d63284cfd4d04bde07ff3351d |
| SHA512 | 512 | 9a0552b9d165360fc08090a88f8c5274ca263c485417c73acb5c1a820b288549... ...12f8885bebbd49f9c229eac9be43441e061408f99e6e25dafaa5c4a946f50693 |

Some common hash functions and their bit sizes

Attacks on Hash Functions

To **brute force** in cryptanalysis is to search the entire space of possible alternatives.

A subset of this is a **dictionary attack** where we throw subsets of the keyspace (*dictionaries*) at the problem.

e.g. cracking UNIX passwords

We can use brute force to attack *pre-image resistance*:

- Say a hash produces an n -bit output: $y = h(x)$
- We must try 2^{n-1} hashes before $\Pr[h(a) = y] \geq 0.5$ ($a \in \mathbb{Z}$)
- Intuitively: if the secret key is one of ($2^{10} = 1024$) boxes, you have to open half of them ($2^9 = 512$) on average before you find the secret key.

Birthday Attacks on CRHFS

A **birthday attack** is an attack on collision resistance:

- How many people must be in a room such that any 2 share a birthday?
 - $\Pr[\text{two people same birthday}] > 0.5?$
 - Requires only 23 people

See: en.wikipedia.org/wiki/Birthday_problem
- For an n -bit hash, we must try $2^{\frac{n}{2}}$ hashes of random messages on average before the attack succeeds.

SHattered - SHA-1 Break

Very recent **February 23, 2017!**

Found by **Google** and **CWI**

<https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>

<http://shattered.io/>

Information

Complexity:

9, 223, 372, 036, 854, 775, 808 SHA-1 compressions!

Nine **QUINTILLION**

Comparisons:

Shattered: 110GPU = 1 year

Brute Force: 12 million GPU = 1 year

What is affected?

Digital Certificates, Email, Software Updates, **GIT**.

Collision Resistance

| Hash Function | Security Claim | Best Attack | Publish Date | Comment |
|---------------|----------------|-----------------------------|--------------------------|---|
| MD5 | 2^{64} | 2^{18} time | 25-03-2013 | This attack takes seconds on a regular PC. Two-block collisions in 2^{18} , single block collisions in 2^{41} |
| SHA-1 | 2^{80} | $2^{60.3} \dots 2^{65.3}$ | 19-06-2012 23-02-2017 | SHAttered was the first public release of a collision. Attack is feasible with large amounts of computation. |
| SHA256 | 2^{128} | 31/64 rounds ($2^{65.5}$) | 28-05-2013 | |
| SHA512 | 2^{256} | 24/80 rounds ($2^{32.5}$) | 25-11-2008 | |

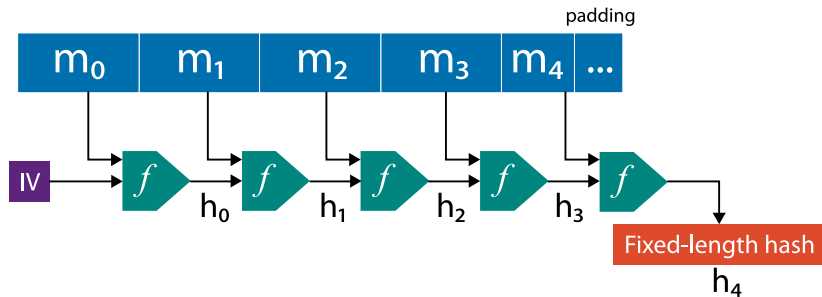
Preimage Resistance

| Hash Function | Security Claim | Best Attack | Publish Date |
|---------------|----------------|--|--------------|
| MD5 | 2^{128} | $2^{123.4}$ | 27-04-2009 |
| SHA-1 | 2^{160} | 45 of 80 rounds | 17-08-2008 |
| SHA256 | 2^{256} | 43 of 64 rounds ($2^{254.9}$ time, 2^6 memory) | 10-12-2009 |
| SHA512 | 2^{512} | 46 of 80 rounds ($2^{511.5}$ time, 2^6 memory) | 25-11-2008 |

en.wikipedia.org/wiki/Hash_function_security_summary

Iterated Hash Construction

Merkle-Damgård construction is a technique for building hash functions.



f is a **one-way compression function**.

Simply divide a message M into n r -bit blocks.

Length must be a multiple of a fixed number (e.g 512 bits), so the message must first pass through a **padding function**.

Why Use Merkle-Damgård?

LEMMA

Suppose the compression function $f(m_n, h)$ is collision resistant.

Then the resulting hash function $h(m)$ is also collision resistant.

To construct a CRHF, it is enough to construct CR compression functions:

$$f : \{0, 1\}^m \times \{0, 1\}^r \rightarrow \{0, 1\}^m$$

Sponge Construction

Some recent CRHFs use [sponge construction](#).

For example, [SHA-3](#).

MD5

| Input | Hash Value (as hex byte string) |
|-------|----------------------------------|
| "" | d41d8cd98f00b204e9800998ecf8427e |
| "a" | 0cc175b9c0f1b6a831c399e269772661 |
| "abc" | 900150983cd24fb0d6963f7d28e17f72 |

SHA-1

| Input | Hash Value (as hex byte string) |
|-------|--|
| "" | da39a3ee5e6b4b0d3255bfef95601890afd80709 |
| "a" | 86f7e437faa5a7fce15d1ddcb9eaeaea377667b8 |
| "abc" | a9993e364706816aba3e25717850c26c9cd0d89d |

MESSAGE AUTHENTICATION CODES (MACs)

Keyed Hash Functions (MACs)

Well known **Message Authentication Codes** (MACs).

A one-way hash function with the addition of a key:

$$h_k : \{0, 1\}^* \rightarrow \{0, 1\}^n$$

The key is secret and necessary to verify the hash $h_k(m)$ and can be thought of as a cryptographic checksum.

GOAL

- Provides message authentication where sender and receiver share a secret.
- An eavesdropper cannot fake a message with a valid MAC.
- Used for message integrity, *not* message **secrecy**.

Properties of MACS

Given m and k it is easy to construct $h_k(m)$.

Given pairs of messages and MACs $(m_i, h_k(m_i))$ it is hard to construct a valid new pair:

$$(m_j, h_k(m_j)) \text{ for } m_j \neq m_i$$

Without knowledge of k .

FORMALLY

A MAC is (ε, t, q, l) .

It is secure if:

Given q pairs, each of length $\leq l$, in time t an adversary can succeed in constructing new (message, MAC) pairs with probability $< \varepsilon$.

Network Example

- Alice and Bob share a secret key \mathbf{k}
- An adversary can't send a message with a valid MAC
 $\text{MAC}(\mathbf{m}) = h_{\mathbf{k}}(\mathbf{m})$

Using MACs - Example 2

Say a hash function is used for virus protection and stores the signatures for each file in a database.

Couldn't the virus also modify the database?

With a MAC, the virus can't because it doesn't know the key.

If it had write permissions, it could however corrupt the database or replace the verification program with a trojan/fake.

Constructing MACs

Cryptographic

- Non-Keyed hash functions (HMAC) - **fast**
- Block cyphers (CBC-MAC) - **slow**

Information Theoretic

- Based on [universal hashing](#) (outside of scope)

Hash Based MAC (HMAC)

MAC based on non-keyed hash function h

Attempt 1: $MAC_k(m) = h(k|m)$

INSECURE: attacker can arbitrarily add to the end of the message. (Merkle-Damgård construction)

Attempt 2: $MAC_k(m) = h(m|k)$

INSECURE: vulnerable to the birthday attack!

Attempt 3: $MAC_{k,k'}(m) = h(k|m|k')$

MORE SECURE: enveloped method

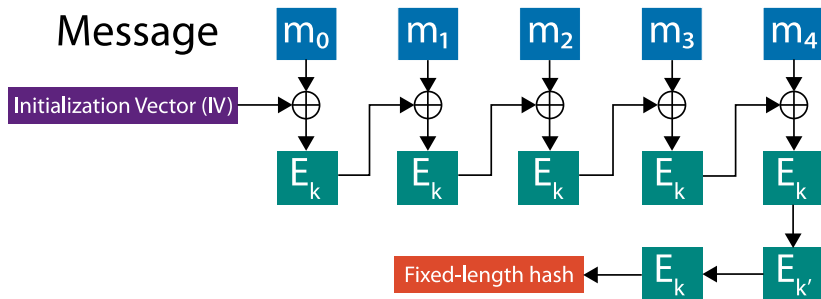
BEST: $MAC_k(m) = h((k \oplus opad)|h((k \oplus ipad)|m))$

- *opad* is the **outer** padding
(0x5c5c5c5... one block of hex constant)
- *ipad* is the **inner** padding
(0x3636363... one block of hex constant)
- from [RFC2104](#)

Cipher-Based MAC (CBC-MAC)

CBC-MAC uses a technique known as **Cipher Block Chaining**.

- Turn a message into blocks
- Repeated encryption using a block cipher is *XORed* (\oplus)
- Secret key = (k, k', IV)
IV: Initialisation Vector (random bits)



If E is a MAC then CBC- E is also a MAC

Often used in the banking industry!

Typical key length

- Very small, e.g. 40 BITS
- Security $\sim 2^{40}$ (*easily guessed*)

No Birthday Attack on MACs

Implies MACs are shorter than message digests!

CBC-MAC Length

| Name | Key Size (bits) | Hash length (bits) | Relative Speed | Class | Notes |
|----------------------|-----------------|--------------------|----------------|---------------|--------------------------------------|
| Blowfish | Up to 448 | 64 | 23 | Block Cipher | Bruce Schneier |
| DES | 56 | 64 | 10.6 | Block Cipher | Lucifer/NSA |
| 3DES | 112 | 64 | 3.7 | Block Cipher | Banking |
| IDEA | 128 | 64 | 11.8 | Block Cipher | Massey and Lai |
| RC5 (r=12) | Up to 2048 | 32, 64, 128 | 19.6 | Block Cipher | Ron Rivest (RSA) |
| AES (r=10, 128 bits) | 128,192,256 | 128,192,256 | 21.1 | Block Cipher | Rijndael |
| CRC32 | - | 32 | 173 | Checksum | Very weak - linear |
| MD4 | - | 128 | 176 | Hash Function | Ron Rivest (RSA) |
| MD5 | - | 128 | 127 | Hash Function | Ron Rivest (RSA) Block collisions |
| SHA-1 | - | 160 | 81.5 | Hash Function | NSA Hash Collisions |

There has been a steady stream of breaks against popular hashing functions like MD5 and SHA-1.

Be sure to pay attention when new hash functions are recommended and when others have been deprecated.

In 2012, [NIST ran a competition](#) to choose the latest generation of hash functions, now known as the [SHA-3 family](#).