Hash Functions

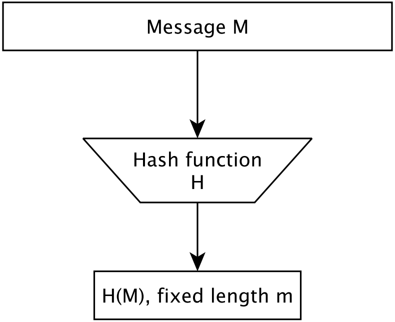
Based on *Applied Cryptography* by Schneier, Chapter 18

Lecture notes of Alexander Wood
awood@jjay.cuny.edu

John Jay College of Criminal Justice

One-Way Hash Functions

A **one-way hash function** $H(M)$ takes as input a message M of arbitrary length. It returns a value h of fixed length m called a **hash value**.



Hash Function Properties: Review

Let H be a one-way hash function and say $H(M) = h$. H must satisfy the following properties:

- Given M it is easy to compute h. In other words, $H(M)$ is a fast operation.
- Given h, it is hard to compute M such that $H(M) = h$.
- Given M, it is hard to find another message M' such that $H(M) = H(M')$.

Collision-Resistance

The property of **collision-resistance** states that it is hard to find *any* two messages M and M' such that $H(M) = H(M')$.

Ideal Hash Functions

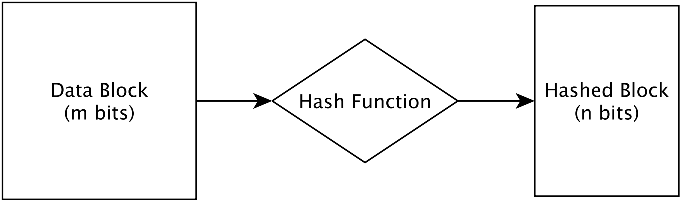
The **ideal hash function** behaves like a random mapping from the set of possible input values to the set of possible output values.

Attacks on Hash Functions

An **attack on a hash function** is a non-generic method of distinguishing between the hash function and an ideal hash function.

Designing Hash Functions

How do we design a function which takes an input of *arbitrary* length? Often we instead design a function which takes a value of a fixed size m and outputs a value of a fixed length n.



Designing Hash Functions

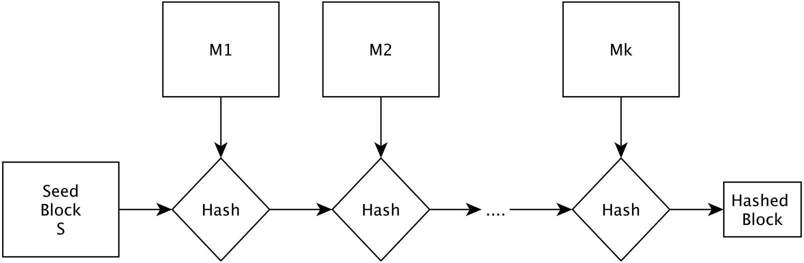
We hash a message by sectioning it into blocks of fixed size m, say $M_1, \ldots, M_k$. Let S be some **seed value** of size m. We compute as follows:

- $H_1 = H(S, M_1)$
- $H_2 = H(H_1, M_2)$
- $\cdots$
- $H_k = H(H_{k-1}, M_k)$

where H_k is the final output, or the hash value.

Sequential Applications of Hashing

The following figure illustrates this concept.



This is known as an **avalance effect**. The hash of the entire message is the hash of the last block.

(Non-)Example 1

First we look at an iterative hash function which is *not* secure to illustrate what we do *not* want.

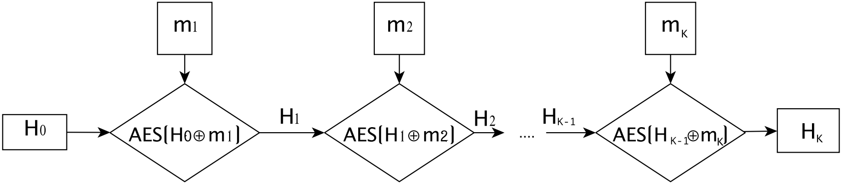
Refer to *Cryptography Engineering* section 5.2.1 for more details on this non-example.

(Non-)Example 1

Let's build a hash function from AES (the Advanced Encryption Standard) with a 256-bit key.

1. Pad the message and break it into 128-bit blocks $m_1, \ldots, m_K$.
2. Set $H_0 := \texttt{00...0}$, a 128-bit block of all zeroes.
3. Compute $H_i = \texttt{AES}_K(H_{i-1} \oplus m_i)$ for $1 \leq i \leq K$.
4. Output hash value H_K.

(Non-)Example 1

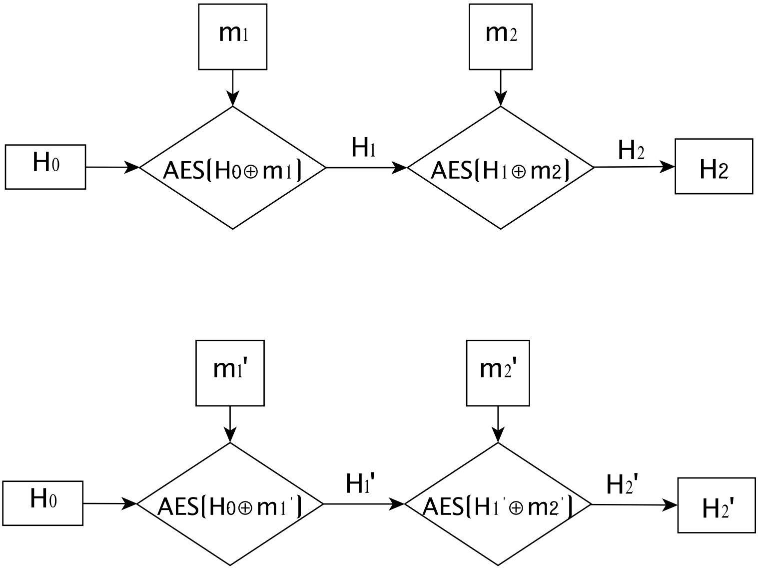


(Non-)Example 1: Attack

Split a message m into two blocks, m_1 and m_2. Let $m'_1 = m_1 \oplus H_1$ and $m'_2 = H_2 \oplus m_2 \oplus H_1$. Let m' be the message that splits into m'_1 and m'_1 after hashing.

What does m' hash to?

(Non-)Example 1: Attack



(Non-)Example 1: Attack

m' hashes as follows:

1) $m'_1 = m_1 \oplus H_1$, $m'_2 = H_2 \oplus m_2 \oplus H_1$
2) $H'_1 = AES(H_0 \oplus m'_1) = AES(H_0 \oplus H_1 \oplus m_1)$
3) Observe that:

$$\begin{aligned} H'_2 &= AES(H'_1 \oplus m'_2) \\ &= AES(AES(H_0 \oplus H_1 \oplus m_1) \oplus H_1 \oplus H_2 \oplus m_2) \\ &= AES(H_2 \oplus H_1 \oplus H_2 \oplus m_2) \\ &= AES(H_1 \oplus m_2) \\ &= H_2 \end{aligned}$$

We found a collision!

MD5

The MD5 algorithm is one of the **Message Digest** functions designed by Ronald Rivest in 1991. It has 128-bit output length and was thought for some time to be collision-resistant.

In 2004 a group of Chinese cryptanalysts demonstrated a method for finding a collision. Now, collisions can be found in under one minute.

SHA-0

The **Secure Hash Algorithm SHA-0**, designed by the NSA, is the first in a series of algorithms standardized by NIST. It was published in 1993.

SHA-0 was withdrawn due to an unspecified weakness the NSA found within the algorithm.

SHA-1

SHA-1 was the successor to SHA-0 in 1995. It has 160-bit output. The birthday bound for this function is only 2^{80} steps; however, it was conjectured that a collision could be found in significantly fewer steps than this.

This conjecture proved correct. Starting in 2005, various cryptologists published their own methods of attack which took well below 2^{80} steps.

The first published collision was found this year, in February 2017! Google generated two different PDF files which have the same SHA-1 hash. It took them approximaetely $2^{63.1}$ evaluations. They called this the SHAttered attack.

SHA-2

SHA-2 was designed by the NSA and published in 2002. It is comprised of six related functions: SHA-SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, which are 224, 256, 384 or 512 bits.

The attacks against SHA-1 have not been successfuly extended to attack SHA-2.

SHA-3

In 2007, NIST announced a competition to design a new cryptographic hash function. NIST selected the Keccack algorithm as the contest winner in 2012 and was standardized by NIST as SHA-3 in August 2015.

Video: Overview of SHA

If interested you may watch the following overview of how SHA works, broadly:

`https://www.youtube.com/watch?v=5OVb4I5fhKI`

Video: SHA-1

If interested you may watch the following video of how SHA1 works:

`https://www.youtube.com/watch?v=aLvwpJcOy6s`

Python: hashlib

We will learn how to implement Python's `hashlib` module.

Please open IDLE on your computers so that we may all implement the following together.

The documentation as well as source code for `hashlib` is available here:
`https://docs.python.org/3/library/hashlib.html`

Python: hashlib

To use hashlib, type

```
>>> import hashlib
```

into the active interpreter.

Python: hashlib

There is one constructor method named for each type of hash. All return a hash object with the same interface.

To see what algorithms we can implement using `hashlib`, type in the following command to your interpreter:

```
>>> hashlib.algorithms_guaranteed
```

Python: hashlib

Let's start with `MD5`. Enter the following commands, one-by-one.

```
md5object = hashlib.md5(b'Hello World')
print(hash_object.hexdigest())
```

The `b` which precedes the string literal converts the string to bytes.

Python: hashlib

Now enter the following, one by one.

```
>>> m = hashlib.md5()
>>> m.update(b"I know when that hotline bling")
>>> m.update(b" that can only mean one thing")
>>> m.digest()
>>> m.digest_size
>>> m.block_size
```

Python: hashlib

```
m = hashlib.md5()
```

Create a md5 hash object.

Python: hashlib

```
>>> m.update(b"I know when that hotline bling")
>>> m.update(b" that can only mean one thing")
```

The `update` method updates the hash object with the argument in the parentheses. The input must be in byte format. It works by appending. For example, `m.update(b'a')` followed by `m.update(b'z')` is equivalent to `m.update(b'az')`.

Python: hashlib

`m.digest()` returns the digest of the data passed to the `update()` method so far. This is a bytes object of size `digest_size` which may contain bytes in the whole range from 0 to 255.

`m.hexdigest()` is like `digest()` except the digest is returned as a string object of double length, containing only hexadecimal digits.

Python: hashlib

```
>>> m.digest_size
>>> m.block_size
```

Return the attributes `digest_size` (size of resulting hash in bytes) and `block_size` (size of internal block size in hashing algorithm).

Python: hashlib

Let's do the same as above with the remaining hash algorithms.

Python: hashlib, SHA-1

Enter the commands one-by-one in the active python interpreter.

```
import hashlib
hash_object = hashlib.sha1(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
blocksize = hash_object.block_size
digestsize = hash_object.digest_size
print(blocksize)
print(digestsize)
```

Python: hashlib, SHA-224

Enter the commands one-by-one in the active python interpreter.

```
import hashlib
hash_object = hashlib.sha224(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
blocksize = hash_object.block_size
digestsize = hash_object.digest_size
print(blocksize)
print(digestsize)
```

Python: hashlib, SHA-256

Enter the commands one-by-one in the active python interpreter.

```
import hashlib
hash_object = hashlib.sha256(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
blocksize = hash_object.block_size
digestsize = hash_object.digest_size
print(blocksize)
print(digestsize)
```

Python: hashlib, SHA-384

Enter the commands one-by-one in the active python interpreter.

```
import hashlib
hash_object = hashlib.sha384(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
blocksize = hash_object.block_size
digestsize = hash_object.digest_size
print(blocksize)
print(digestsize)
```

Python: hashlib, SHA-512

Enter the commands one-by-one in the active python interpreter.

```
import hashlib
hash_object = hashlib.sha512(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
blocksize = hash_object.block_size
digestsize = hash_object.digest_size
print(blocksize)
print(digestsize)
```

References

- *Applied Cryptography* By Schneier, Chapter 18
- The HashLib documentation: `https://docs.python.org/3/library/hashlib.html`
- Python implementations: `http://pythoncentral.io/hashing-strings-with-python/`