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

Hashing

A **hash function**, informally, is a method of taking a numerical input of one value and returning a compressed numerical input of a fixed (smaller) length.

Why Hash, generally?

In modern days, there is truly a massive amount of content being generated by users all over the world every day.

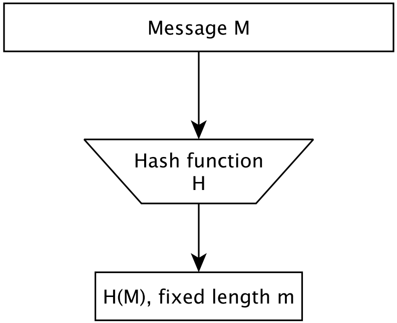
Simply in terms of data storage, why will traditional data structures such as arrays and linked lists no longer operate efficiently?

Why Hash, generally?

These methods of data storage can take data retrieval time of linear order $\mathcal{O}(n)$. By **hashing** the values we can retrieve data in only linear time $\mathcal{O}(1)$.

Why Hash... In Cryptography?

Cryptographic hash functions are a main feature of modern cryptography.



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**.

Vocab!

- The process of converting data from an arbitrary length to a fixed length is called **hashing** the data.
- Hash functions are sometimes referred to as **compression functions**, since the fixed length output is in most applications smaller than the input values.
- The hash of a large data value is referred to as a **digest**.
- If the fixed length of the output of a hash function is m bits, we call H an m**-bit hash function**.

Hash Function Properties

A one-way hash function 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 $H(M) = H(M')$.

Pre-Image Resistance

The second property in the previous slide is called **pre-image resistance**. If $H(M) = h$, An adversary should not be able to recover a value M.

While this is usually not a strong enough notion of resistance for cryptographic applications, it is an important baseline. Pre-image resistance makes it so that an adversary Eve should have computational difficulty finding an input for any hash value she might intercept.

Second Pre-Image Resistance

The last property in the Hash Function Properties slide is referred to as **second pre-image resistance**. Say $H(M) = h$, and Eve intercepts h. It should be computationally difficult for Eve to find a message M such that $H(M') = h$.

Again, we need a stronger notion of resistance in most cryptographic applications.

Collisions

Say Alice signs M with a digital signature $H(M)$. If Eve could produce M' such that $H(M') = H(M)$, then Eve could forge Alice's signature.

Collision-Resistance

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

The key observation now is that a malicious party can construct their own malicious hash values for which a collision has been found. Let's consider an example to see how dangerous this really could be.

Digital Signatures

A **digital signature** scheme provides a mathematical method for demonstrating the authenticity of a message M. It must satisfy the properties of **authentication**, **non-repudiation**, and **integrity**.

Digital signatures were first described in 1976 by Whitfield Diffie and Martin Hellman.

Digital Signatures

A digital signature protocol consists of three algorithms:

- Key Generation Algorithm
- Signing Algorithm
- Signature Verification Algorithm

Birthday Attacks

Recall we discussed birthday attacks carried out against a digital signature.

If Alice wishes to carry out a Birthday Attack to forge a signature from Bob, then she wishes to find two random messages M and M' such that $H(M) = H(M')$.

Birthday Attacks: Example Protocol

Say we *don't* have a collision-resistant hash function. Consider the following protocol, described by Gideon Yuval:

- Alice prepares two versions of a contract: one fair contract C, and one which bankrupts Bob, $\hat{C}$.
- Alice makes several subtle changes to each contract (ie, replace space with space-backspace-space) and calculates the hash value for each.
- Alice continues making subtle changes to each document and generating hash values until she finds a pair of matching hash values. In other words, she finds versions of C and $\hat{C}$ such that

$$H(C) = H(\hat{C}).$$

Example Protocol, Continued.

- Bob signs the fair contract, C.
- Alice replaces the fair contract with the malicious one, $\hat{C}$. Now she can convince an adjudicator that Bob has signed $\hat{C}$, since the hash values of both documents are the same.

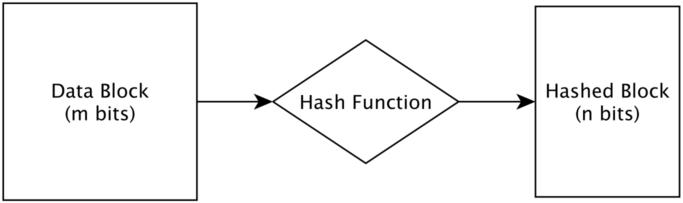
Another Example

Another example of an attack provided by Schneier is that of hacking an enemy satellite with an automated control system. Eve could send the satillite random messages with random signature strings. While most of these signature strings will not be valid, assuming a successful Birthday Attack can take place, eventually one of them may be validated.

While Eve is not in control of the random instruction she sent to the satellite, the idea is that if her only objective was to tamper with the satellite, then any random instruction is a success.

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

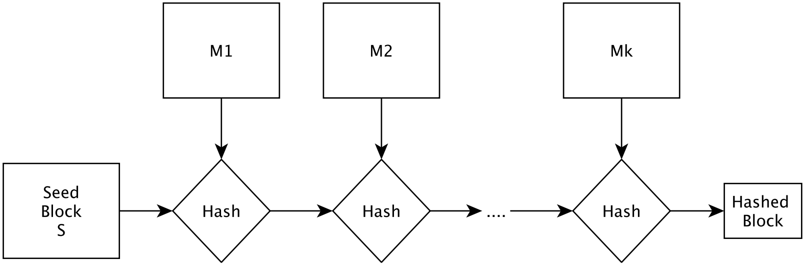
We hash a message by sectioning it into blocks of fixed size m, say $M_1, \ldots, M_k$. Let S be some **seed value** of size m. We compute as follows:

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

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.

Popular Hash Functions

Will be discussed in more detail next week.

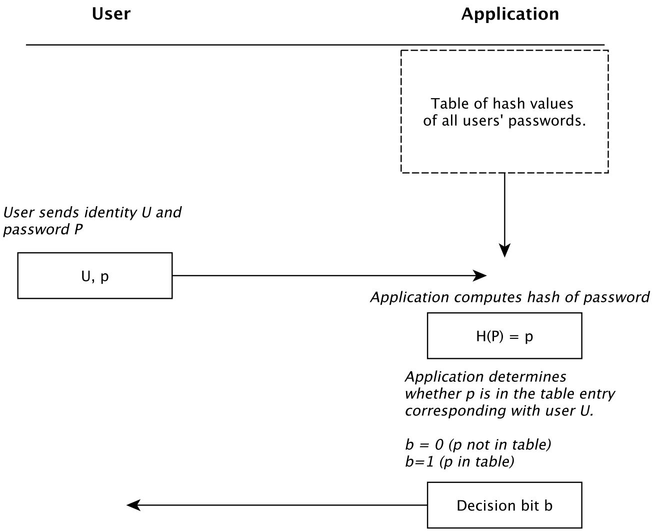
- Message Digest (MD)
- Secure Hash Function (SHA)
- RIPEMD
- Whirlpool

Hash Values for Password Storage

Often times, the cleartext value of your password is not what will be stored by a logon server. Instead, the hash of your password is stored. Thus a malicious third party who hacks the database only has access to the hash of your password – not the password itself.

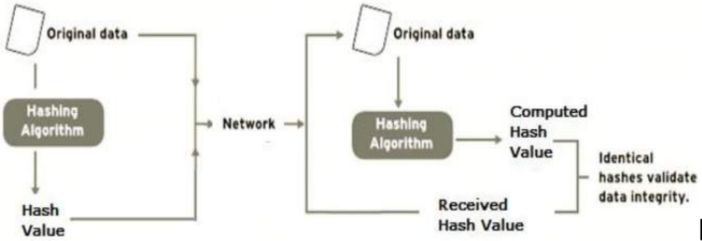
Due to collision-resistance, this malicious third party should not be able to forge your credentials and long on in your name.

Password Storage Diagram


User
Application
Table of hash values of all users' passwords.
User sends identity U and password P
U, p
Application computes hash of password
H(P) = p
Application determines whether p is in the table entry corresponding with user U.
b = 0 (p not in table)
b=1 (p in table)
Decision bit b

Hash Functions for Data Integrity

Suppose you wish to verify that someone has not modified an original file.



Note that the verifier must know independently about the originality of the data.

Overview of SHA

We now watch the following video for a broad overview of SHA.
`https://www.youtube.com/watch?v=5OVb4I5fhKI`

SHA-1

Let's watch the following video of how SHA1 works.
`https://www.youtube.com/watch?v=aLvwpJcOy6s`

References

- *Applied Cryptography* By Schneier, Chapter 18
- https://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture17.pdf
- https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm