Unit 2: Cryptographic Hashing

SI 486I: Randomized and Blockchain Technology

Assoc. Prof. Daniel S. Roche
United States Naval Academy

Spring 2022

Goals of this unit

- Know what a cryptographic hash function is
- Understand the random oracle model of hash functions, why it is useful and what are its limitations
- Understand the three security properties needed for blockchains: collision resistance, hiding, and puzzle-friendliness
- Be able to use these security properties and reason about them
- Know the basic structure of a cryptographic sponge
- Follow the main steps of Keccak hashing for SHA3

Cryptographic primitives

Primitives are building blocks for crypto protocols.

Examples:

- Hashing
- Signatures
- Encryption

Big cryptographic principles

These are the law!

- **Don't roll your own crypto** (primitives)
- **The algorithm is not a secret** (Kerckhoffs's principle)
- **Assume attackers are smart, patient, and extremely wealthy**

Hash function definition

A hash function:

- Takes any amount of input (typically a stream of bytes)
- Produces a fixed-length output (e.g., 32 bytes)
- Is fast to evaluate

Colorful analogy: sausage mixing

Better analogy: (pocketless) billiards

Algorithm speeds for this class

- Fast
- Slow
- Infeasible

Probabilities for this class

- Likely
- Unlikely
- "Impossible"

Desired hash function properties

Assume H is a hash function with n-bit outputs.

Collision resistance
No one will ever find $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$

Hiding
Given a "random" y, it is infeasible to find x such that $H(x) = y$

Puzzle friendliness
Given a subset Y of size $m = \#Y$ of hash digests,
and a "random" prefix r,
finding an x so that $H(r \| x) \in Y$ takes $O(2^n/m)$ time

Thought experiment: combining hash functions

Imagine we have two hash functions H_1 and H_2. One of them is weak but we're not sure which.
We want to combine them into a single strong hash function H_3

Which way(s) will work and which will fail?

- Chaining: $H_3(x) = H_1(H_2(x))$
- Concatenation: $H_3(x) = H_1(x) \parallel H_2(x)$
- Bitwise AND: $H_3(x) = H_1(x)$ & $H_2(x)$

Random Oracle Model

Goal: Help us think about how to use strong hash functions

Santa Claus Hash

- To compute $H(x)$, send x to Santa
- Santa checks his list:
  - If x is on the list, he returns the listed answer
  - Otherwise, he computes a new, random answer, adds it to the list for the future, and returns it.
- Important: Santa is omnipresent and has perfect memory. (The list contains all previously-hashed values by anyone.)

Note: controversial; subject of serious cryptographic flame wars

Using ROM

Assume H is a random oracle with 256-bit outputs.

How do you know H is:

- Collision resistant?
- Hiding?
- Puzzle friendly?

Hashing applications

Hash function standards

- MD5 (Ron Rivest), 1992
- RIPEMD-160 (Belgium), 1992
- SHA-0 (NSA), 1993
- SHA-1 (NSA), 1995
- SHA-2 (NSA), 2001
- BLAKE (Switzerland), 2008
- Keccak (Netherlands), 2008

Hash function designs

- Merkle-Damgard construction
  Used in MD5, RIPEMD, SHA-1, SHA-2
- Sponge construction
  Used in Keccak (SHA-3)
- Others: HAIFA, Unique Block Iteration, ...

SHA3-256 details

(aka Keccak[r=1088, c=512])