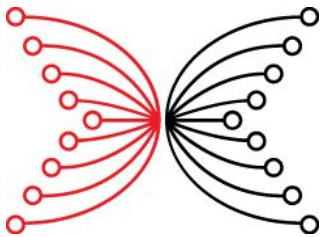
Lesson 3
Hashing

A. Sorokin (Input | Output)

Self paced course

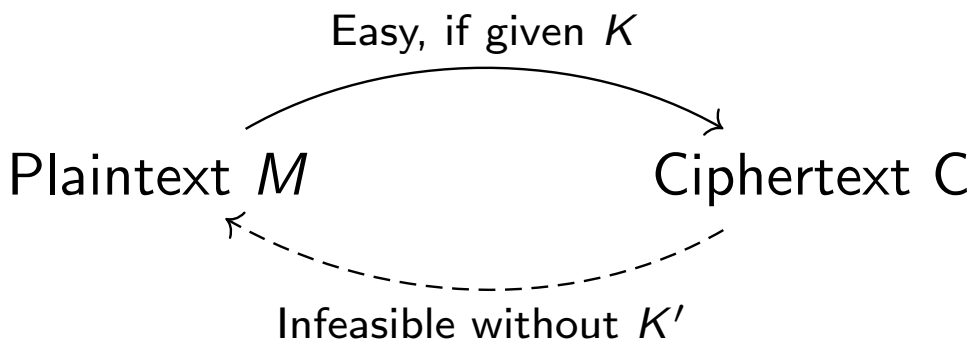
July 31, 2025



Motivation

In public-key cryptosystems an encryption algorithm is designed in a way that without knowing a private key nobody can break the security:

posession of ciphertext provides no information about the plaintext.



Such property is based on *one-way functions* – functions whose action cannot be reversed.

One-way functions

Definition

A function $y = f(x)$ is said to be **one-way** if the following two properties hold:

(i) *Easy-to-compute*: given x, the value $y = f(x)$ can be computed effectively;

(ii) *Hard-to-invert*: there is no known effective algorithms (for the most of y's) to find x such that $y = f(x)$, if we know y.

Even if some number of pairs (x, y) such that $y = f(x)$ is known, it is still hard to invert f.

Existence of one-way functions

While talking about one-way functions it is important to keep in mind that:

(i) *no function has been proved to be one-way*;

(ii) some functions are *believed* to be one-way.

Hash functions

Among one-way functions there are functions with a fixed output size k (in bits). Such functions are called **hash functions**:

$$h : \{\text{string of bits of any length}\} \longrightarrow \{\text{string of bits of length } k\}$$

The values $h(x)$ of a hash function h are called **hashes** (or **digests**).

Example

Here are examples of digests for SHA-256 (a member of the SHA-2 family):

SHA-256("")	=	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
SHA-256("Hello")	=	185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
SHA-256("00000")	=	dfc7f3a01aa357a0c10311028653e8f4d36f1a01661a9b45f4782e8eb0acf2c4
SHA-256("00001")	=	a4e624d686e03ed2767c0abd85c14426b0b1157d2ce81d27bb4fe4f6f01d688a

Digests for SHA-256 have 256 bits (or 64 hexdecimal digits) length.

Important hash functions

Modern hash functions that are (were) used in practice:

- MD5, SHA-1 (both outdated)
- SHA-2 (Bitcoin), SHA-3
- BLAKE2, BLAKE3 (blockchains)
- Whirpool (file integrity verification)
- RIPEMD-160 (was used for Bitcoin)
- Argon2, bcrypt (password storages)

Required properties

The most important and widely used cryptographic hash functions satisfy the following properties:

(1) **uniformity** – every digest appears with approximately the same probability;

(2) **collision resistance** – its hard to find a pair of inputs with the same digest;

(3) **avalanche effect** – any small change to an input leads to unpredictable change to a digest.

Furthermore, in practical applications hash functions are computed a huge number of times, so an algorithm computing digests should be fast.

Number of digests

Usually, the length of a hash function output k is such that the number of all possible digests 2^k is extremely large.

For example, digests of SHA-256 have length 256, so there are

$$2^{256} \approx 1.1579 \times 10^{7}7$$

possible digests for SHA-256. This number is just 1000 times less than the number of atoms in the observable universe.

In practice, for a hash function h being uniform and collision resistant means that inputs and their digests are in one-to-one correspondence with each other, so *digests can serve as unique identifiers of objects*.

Practical applications: data integrity and storage

(a) Hash functions are used to ensure that files, downloads, or transmitted data haven't been altered.

How it works:

(i) A sender computes the hash of a file and shares it with the recipient.

(ii) The recipient recomputes the hash and compares it to the original.

(iii) If there is a mismatch, then file was corrupted or tempered.

(b) Moreover, hashing prevents duplication and is used for elimination of duplicate files (e.g., cloud storage, backups, version control):

$$\textit{identical hashes} \Longrightarrow \textit{duplicate files}$$

Practical applications: data identification and lookup

(c) Hashing allows to identify files uniquely and create a unique fingerprint.

(d) Using hashes of files/entries as keys for arrays accelerates search operations in storages/databases.

Practical applications: password storage

Secure storing passwords without revealing the plaintext is realized via hashing.

How it works:

(i) Passwords with an added random string (called salt) are hashed and stored in a database as a pair

$$\text{(hash-of-salted-password, salt)}$$

(ii) During login, the system hashes the input + salt and compares it to the stored hash.

(iii) In a case of mismatch login is failed.

Even if the database is breached, attackers can't easily recover passwords.

Practical applications: digital signatures and certificates

Hashes are used for verifying the authenticity of messages and software.

How it works:

(i) A hash of the message is computed.

(ii) A signature is the result of encryption of the hash with a private key.

(iii) Recipients decrypt it with the public key and verify if the hash matches.

Practical applications: blockchain and cryptocurrencies

In blockchains hash functions are used for many purposes:

(a) in proof-of-work algorithms as a puzzle that miner solve;

(b) for transaction IDs by hashing transaction data;

(c) to keep the integrity of the whole blockchain;

(d) for preventing spam/DDoS attacks by requiring computational work.

Practical applications: pseudorandom number generators

Hash functions are used for creating pseudorandom numbers for cryptography:

> *A seed is hashed repeatedly to generate random-looking output*.