# Bitcoin and Cryptocurrency Technologies
# Lecture 2: Cryptography Basics 1/2

Yuri Zhykin

Feb 15, 2022

# Introduction to Cryptography

- **Cryptography** (Ancient Greek, "hidden, secret" and "to write"), is the practice and study of techniques for *secure communication* in the *presence of **third parties** called adversaries*.
- **Modern cryptography** is heavily based on *mathematical theory and computer science practice*.
- **Modern cryptographic algorithms** are designed around computational hardness assumptions.

# Introduction to Cryptography 2/2

- Modern cryptography is divided into two categories:
  - **symmetric cryptography** - both parties share the same secret key, used for both encryption and decryption,
  - **asymmetric (public-key) cryptography** - key consists of public and private components; public key is used for encryption, private key - for decryption.
- Cryptography protocols serve two main purposes:
  - **concealing communication** (encryption/decryption),
  - **ensuring integrity of communication** (signing/signature verification)

# Symmetric Cryptography

- The only type of cryptography until 1976.
- Both parties have a shared secret key that is used for both encryption and decryption.
- Symmetric encryption alrgorithms are very fast (e.g. *AES*, *Salsa20*, *ChaCha*).
- Most popular symmetric encryption algorithms are implemented in hardware (e.g. *AES* and AES-NI instruction set for x86 CPUs).
- Perfect encryption scheme:

$$E = M \oplus K,$$
$$D = E \oplus K,$$
$$|M| == |K|$$

# Symmetic Cryptography Problems

- Secret key must be shared beforehand over a secure communication channel - "chicken-and-egg" problem.
- Symmetry of failure - if any of the parties leaks the key, both parties are compromised.
- If multiple parties share the same key, the symmetry of failure affects all parties.
- If each party keeps a different key for each other party (ideally), key storage is yet another problem.

# Public-key Cryptography

- Groundbreaking discovery by **Whitfield Diffie** and **Martin Hellman** in 1976.
- Messages are encrypted with public key, but can only be decrypted with the private key.
- Each party is responsible only for its own private key - public key can be derived from it, if lost, and can be exchanged over insecure communication channels

# Probability, Randomness and Large Numbers 1/3

- In most modern cryptographic systems, the security of the keys is based on probability of guessing very large numbers.
- In order to make this as hard as possible, the numbers must be truly random, i.e. the probability of each bit in the number to be either 1 or 0 must be 0.5.
- Probability of guessing a truly random number $N$ is exactly $1/N$.
- The estimated number of atoms in the Universe is $10^{78} \simeq 2^{259}$, so guessing a 256-bit key is almost equivalent to guessing a particular atom in the Universe.

- At the same time, large numbers used as keys can be compactly represented with hexadecimal (base-16) or base-64 encodings:

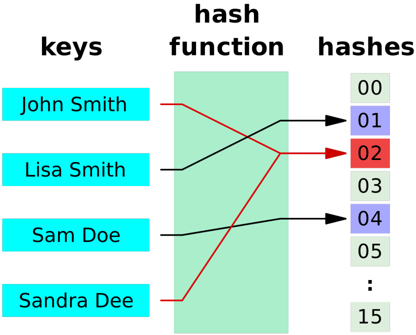
```
import secrets
import base64
bits = secrets.randbits(256)
# 46518555179467323509970270980993648640987722172281263586388328188640792550961
bits_binary = '{0:b}'.format(bits)
# 1100110110110001001000110110101111011010111111101010001111001010000001000100101\
# 1111001101010011111101011111001001110001011101010111000110010101110010110000001\
# 1110101101010100000010001000001111110111110011000000110011100100111111010110100\
# 1001000011110001100001
bits_hex = hex(bits)
# 0x66d891b5ed7f51e5044be6a7ebe4e2eae32b960f5aa0883f7cc0ce4fd6921e31
bits_base64 = base64.b64encode(bits.to_bytes(32, 'little'))
# MR6S1k/OwHw/iKBaD5Yr4+ri5Oun5ksE5VF/7bWR2GY=
```

# Probability, Randomness and Large Numbers 3/3

- Computers are deterministic, so generating truly random numbers is hard.
- In most cases, the most secure way to get a random number on Unix systems is to read from `/dev/random` or `/dev/urandom` devices.
- Special hardware exists for generating truly random numbers based on environment entropy; they should be preferred if possible.

# Hash Functions

- **Hash function** is a function that maps data of arbitrary size to fixed-size values.
- Hash functions are used for storage adressing (hash tables), probabilistic filtering (bloom filters), etc.



# Cryptographic Hash Functions

- **Cryptographic hash function** is a *hash function* that maps data to bit arrays of fixed size and is a **one-way function**, that is, a function that is practically infeasible to invert:

$$h = H(m) \text{ - efficient,}$$
$$m = H^{-1}(h) \text{ - } \textbf{very} \text{ inefficient}$$

- The **most efficient** way to find a message $m$ that produces a given hash $h$ is a **brute-force search** - generate random messages $m_i$ and check if $H(m_i) = h$.
- Basic tool of modern cryptography.

# Properties of Cryptographic Hash Functions

- Main properties of **good** cryptographic hash functions:
  - **determinism** - same input always produces same output,
  - **efficiency** - hash of a given message can be computed quickly,
  - **diffusion, "avalanche effect"** - a single-bit change in $m$ causes change of every bit in $h$ with probability 0.5,
  - **pre-image resistance** - given hash $h$, it should be hard to find any message $m$ such that $h = H(m)$,
  - **second pre-image resistance** - given input $m_1$, it should be hard to find any $m_2$ such that $H(m_1) = H(m_2)$,
  - **collision resistance** - it should be hard to find any messages $m_1$ and $m_2$ such that $H(m_1) = H(m_2)$.
- Additionally:
  - **length extension resistance** - given $h = H(m)$ and $len(m)$, it should be hard to find $h' = H(m||m')$,
  - **strong collision resistance** - **birthday attack** resistance.

# Use of Cryptographic Hash Functions 1/2

- **Message authentication codes (MACs)** - hash of some message combined with some key allows to verify the integrity of the message.
- **Digital signatures** - signing the hash of a message is much more efficient than the signing the whole message.
- **Password verification** - storing cleartext passwords will cause a massive security breach when the database gets leaked; storing password hashes solves this.
- **Strong data integrity checks (checksums)** - used instead of regular (non-cryptographic) hash-functions when stronger guarantees are needed.
- Notable examples: **SHA-2** (**SHA-256**, **SHA-512**), **RIPEMD-160**, **SHA-3**.

# Use of Cryptographic Hash Functions 2/2

- **Proof-of-Work** - basis of modern cryptocurrency technology.
- *Hashcash* - originally proposed by Adam Back in 1997 as means to mitigate email spam and denial of service attacks.
- Basic idea behind **PoW**:
  - For some message $m$, execute **brute force search** on $r$ value until $h = H(m, r)$ meets certain criteria, for example

$$h < h_{target}$$

  - Search criteria can be selected in a way that ensures that with the current state of chip manufacturing, this computations on average takes a certain amount of time.
- This construction essentially means that computing a **PoW** solution requires a provable amount of energy, which can be made sufficiently large to make counterfeiting infeasible.

# Useful Resources

- Dan Boneh's Cryptography I course from Stanford University - https://www.coursera.org/learn/crypto.
- Serious Cryptography: A Practical Introduction to Modern Encryption - Jean-Philippe Aumasson.
- 8 sets of cryptography problems that introduce various real-life cryptography systems and show practical attacks on them - https://cryptopals.com.

# The End

Thank you!