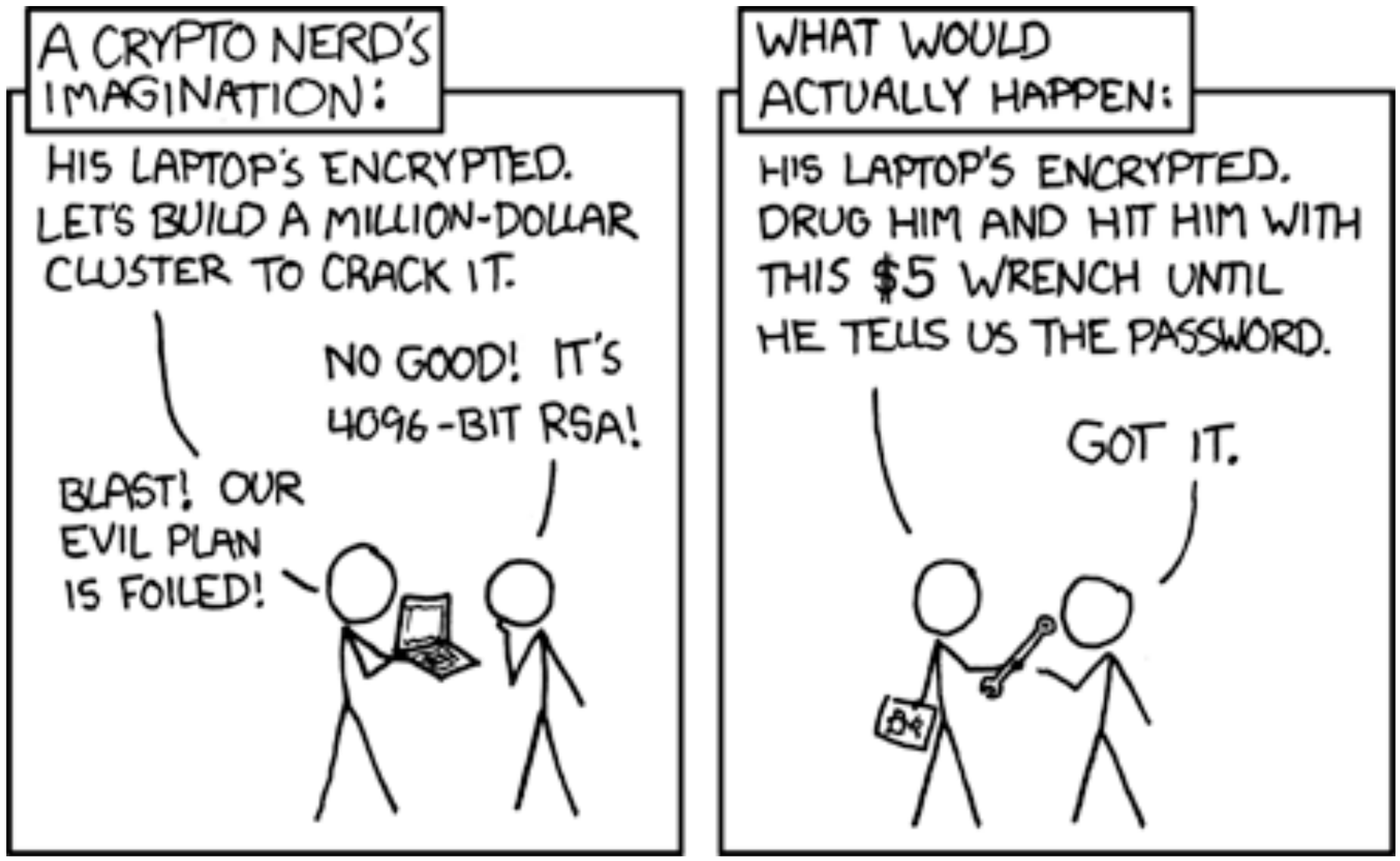
Hash Functions

February 8, 2022

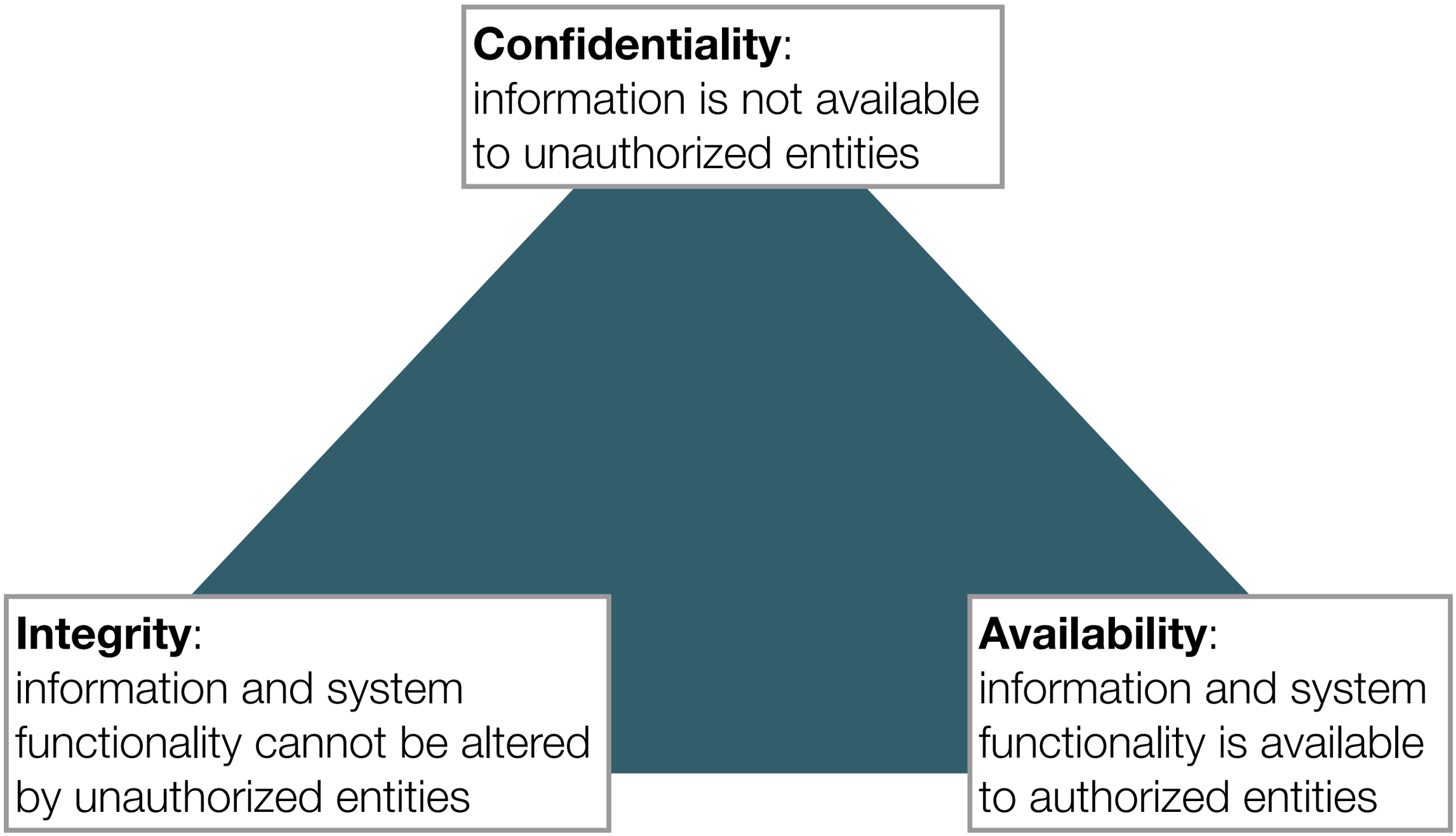


Homework Assignment & Today

- Homework 1
  - available on **Blackboard**
  - based on cryptography lectures, **requires Python or Java programming**
  - due **February 20th** (Sunday) at 11:59pm
- Today: **integrity**
  hash functions
  - *What are hash functions?*
  - MD5, SHA-1,2,3

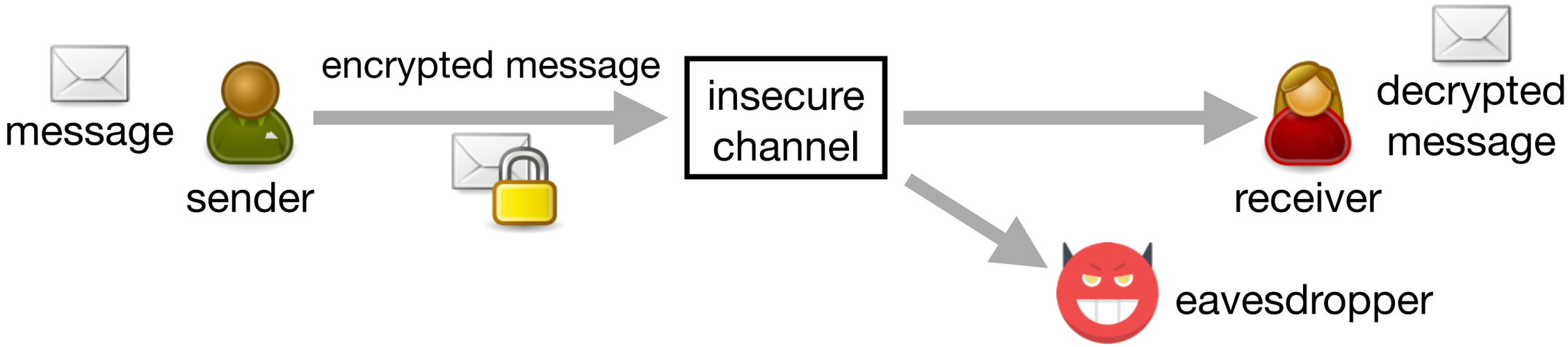
Feedback: https://forms.gle/JGbNCmCsU69iWaTv8

Reminder:
Security Objectives



Attacks Against Communication

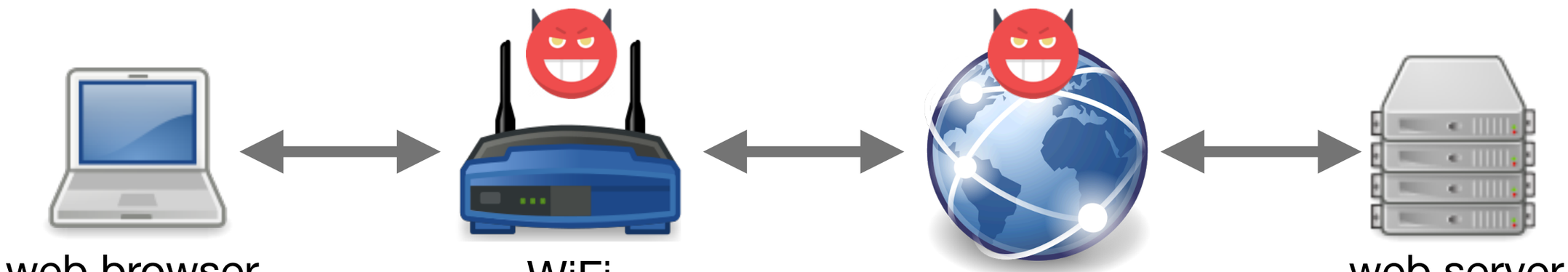
- *Previously*: protecting **confidentiality** against **passive attacks**



- Active attacks

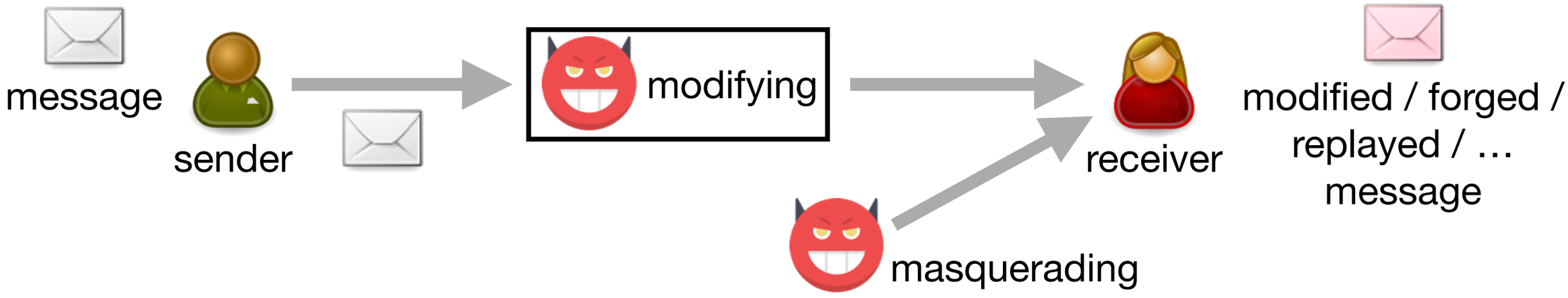
Data integrity: information cannot be **modified in an unauthorized and undetected manner**

Man-in-the-Middle Attack Example



- Data integrity: information cannot be modified in an unauthorized and **undetected** manner
  - very often, preventing unauthorized modifications is impossible, so we must settle for detection

Active Attacks in a Communication Channel



- **Content modification**: changing the contents of a message
- **Sequence modification**: changing the sequence of messages, including deleting some of them
- **Timing modification**: delay or replay messages
- **Masquerade** (i.e., forgery): inserting messages of fraudulent source

Reminder:
Tampering with Stream Ciphers

Original plaintext:	`Y`	`E`	`S`
binary representation:	`01011001`	`01000101`	`01010011`
Pseudorandom sequence: (example)	⊕ `11010010`	`00100000`	`11110101`
Original ciphertext:	`10001011`	`01100101`	`10100110`

Modified ciphertext: `10011100` `01101111` `11010100`

Pseudorandom sequence:	⊕ `11010010`	`00100000`	`11110101`
	`01001110`	`01001111`	`00100001`

Modified plaintext: `N` `O` `!`

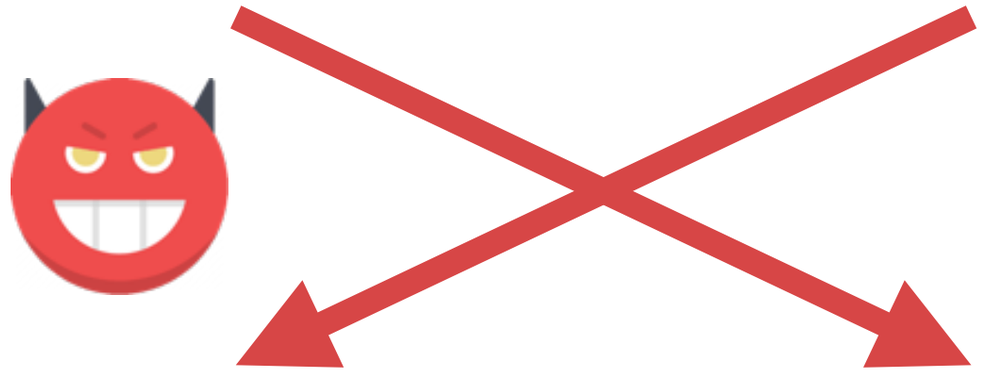
Reminder:
Tampering with Block Ciphers

Original plaintext

Transfer one	million USD to	John Smith's	account from	John Doe's	account.

Original ciphertext

dgyACJVKcERNl	z9iIcfkeBEYE2	sp1uELybLi3wm	fq6aSDNIa6wn6	5YRnb75iDRSFx	wFR0yVk1UrIx0



Modified ciphertext

dgyACJVKcERNl	z9iIcfkeBEYE2	5YRnb75iDRSFx	fq6aSDNIa6wn6	sp1uELybLi3wm	wFR0yVk1UrIx0

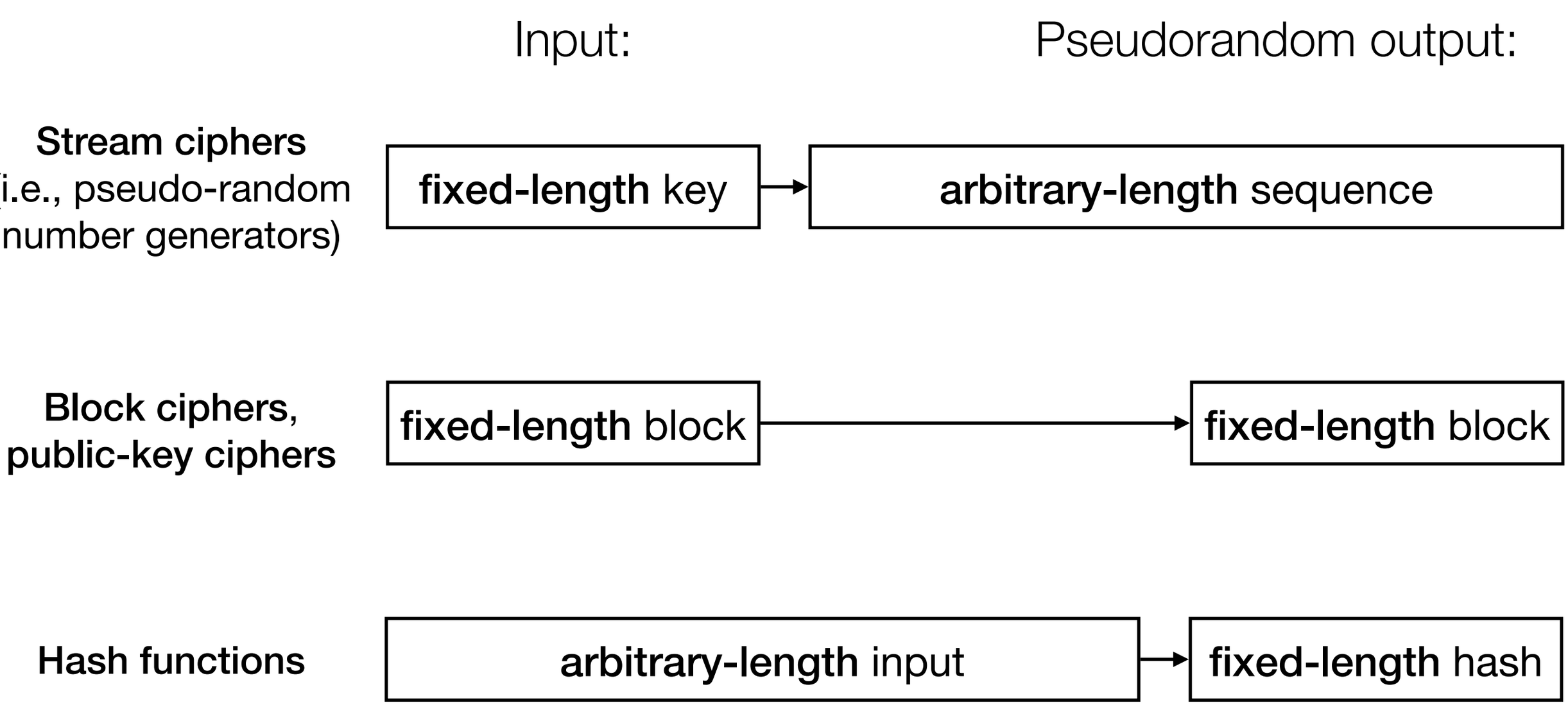
Modified plaintext

Transfer one	million USD to	John Doe's	account from	John Smith's	account.

Integrity

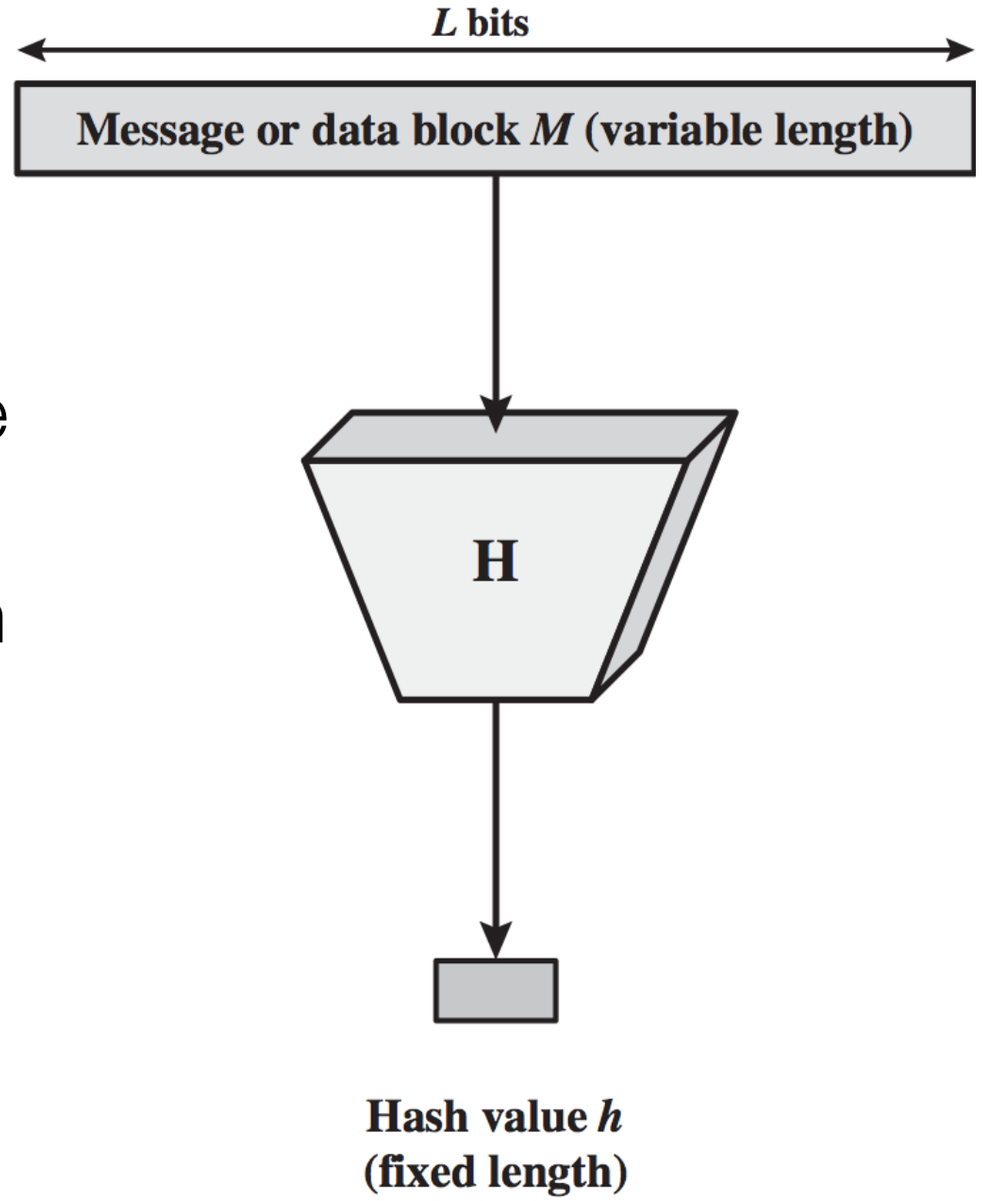
Hash Functions

Hash Functions vs. Other Crypto Primitives



Cryptographic Hash Functions

- **Hash function H**:
  deterministically maps an input **M**
  to a fixed-length hash value **H(M)**
- Requirements
  - **efficient**: computing the hash value of a given input is easy
  - **one-way**: finding an input for which the output is a given hash value is hard (i.e., function is hard to invert)



Hash Function Application
Password File

- Suppose that we store the users' passwords on a server

```
user   :password:userID:groupID:name
-------------------------------------
fry    :abc123   1001   :1000    :Philip...
leela  :p455w0rd 1002   :1000    :Turanga...
bender :qwerty   1003   :1000    :Bender...
```

  - if a hacker compromises the server, it will learn all the passwords

- ~~Encrypting passwords~~
  - secret key would also have to be stored on the server (or be accessible to the server)
    → hacker could easily decrypt the passwords

Hash Function Application
Password File (contd.)

- Store only the **hash values of the users' passwords** on the server
  - when a user tries to log in, we compute the hash of the password entered by the user, and compare it with the stored hash value

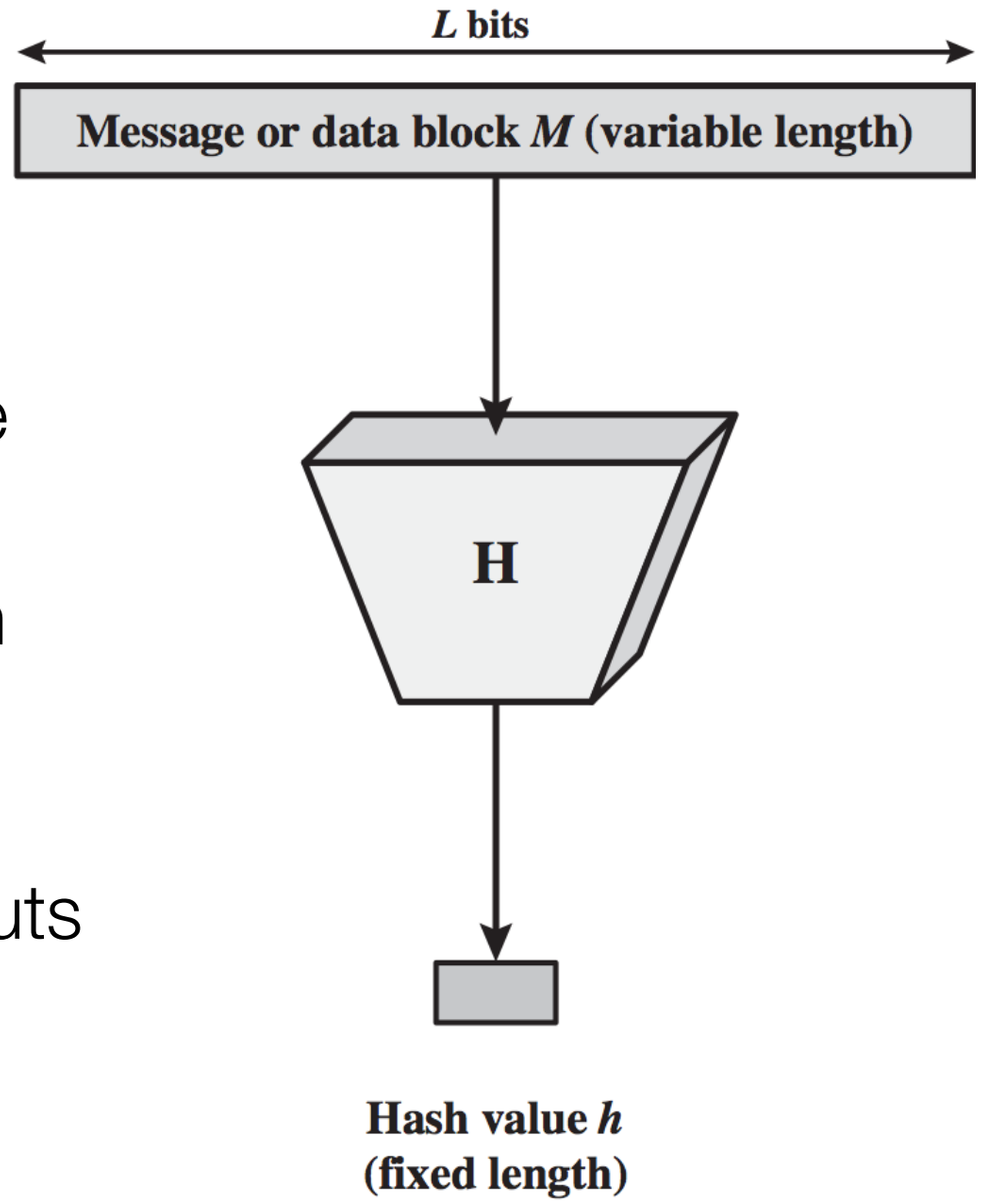
```
user   :password (hash):userID:groupID:name
-------------------------------------------
fry     708eb906206fd92 1001   :1000    :Philip...
leela   32aa6e18b680faa 1002   :1000    :Turanga...
bender  8de40d30c73e6fb 1003   :1000    :Bender...
```

  - if a hacker compromises the server, it will learn only the hash values but not the actual passwords

- **One-way hash function**:
  it is hard to find a password whose hash is a given value

Cryptographic Hash Functions

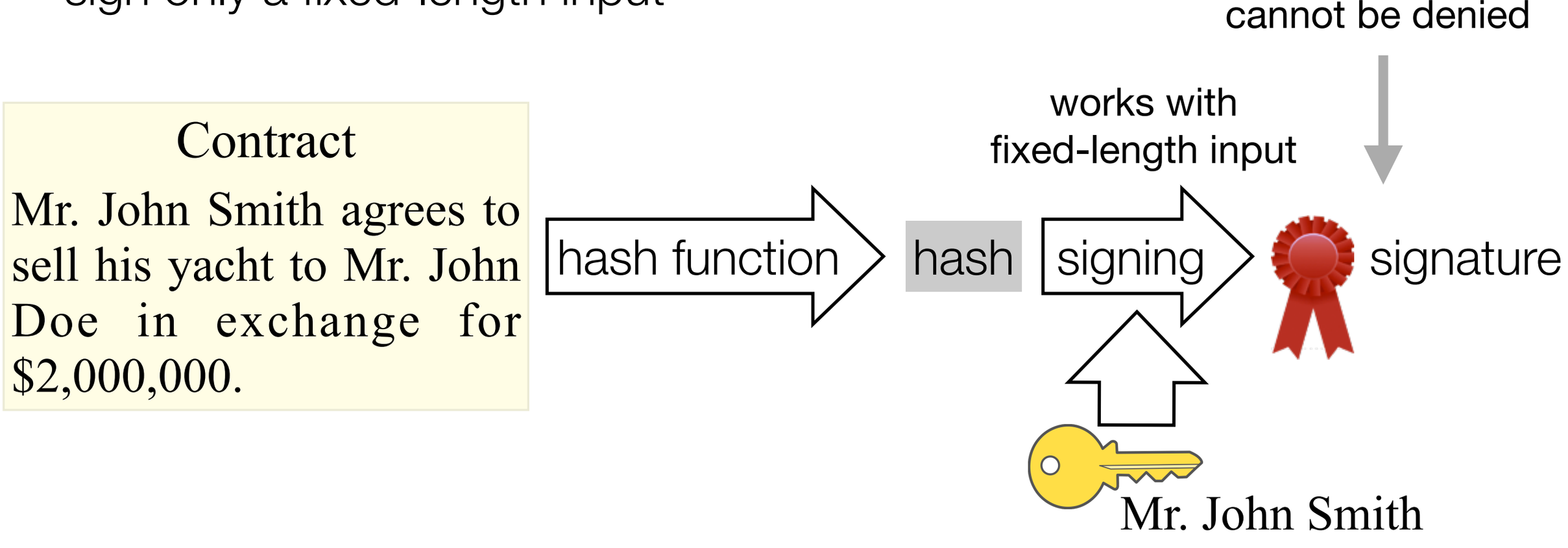
- **Hash function H**:
  deterministically maps an input **M**
  to a fixed-length hash value **H(M)**
- Requirements
  - **efficient**: computing the hash value of a given input is easy
  - **one-way**: finding an input for which the output is a given hash value is hard (i.e., function is hard to invert)
  - **collision-resistant**: finding two inputs for which the hash values are the same is hard



Hash Function Application
Digital Signatures

- **Digital signature**: functions similarly to traditional signatures
  - signatures cannot be forged
  - signee cannot deny signing a document
- *Problem*: most digital signature algorithms can sign only a fixed-length input

cannot be forged,
cannot be denied



Hash Function Application
Digital Signatures (contd.)

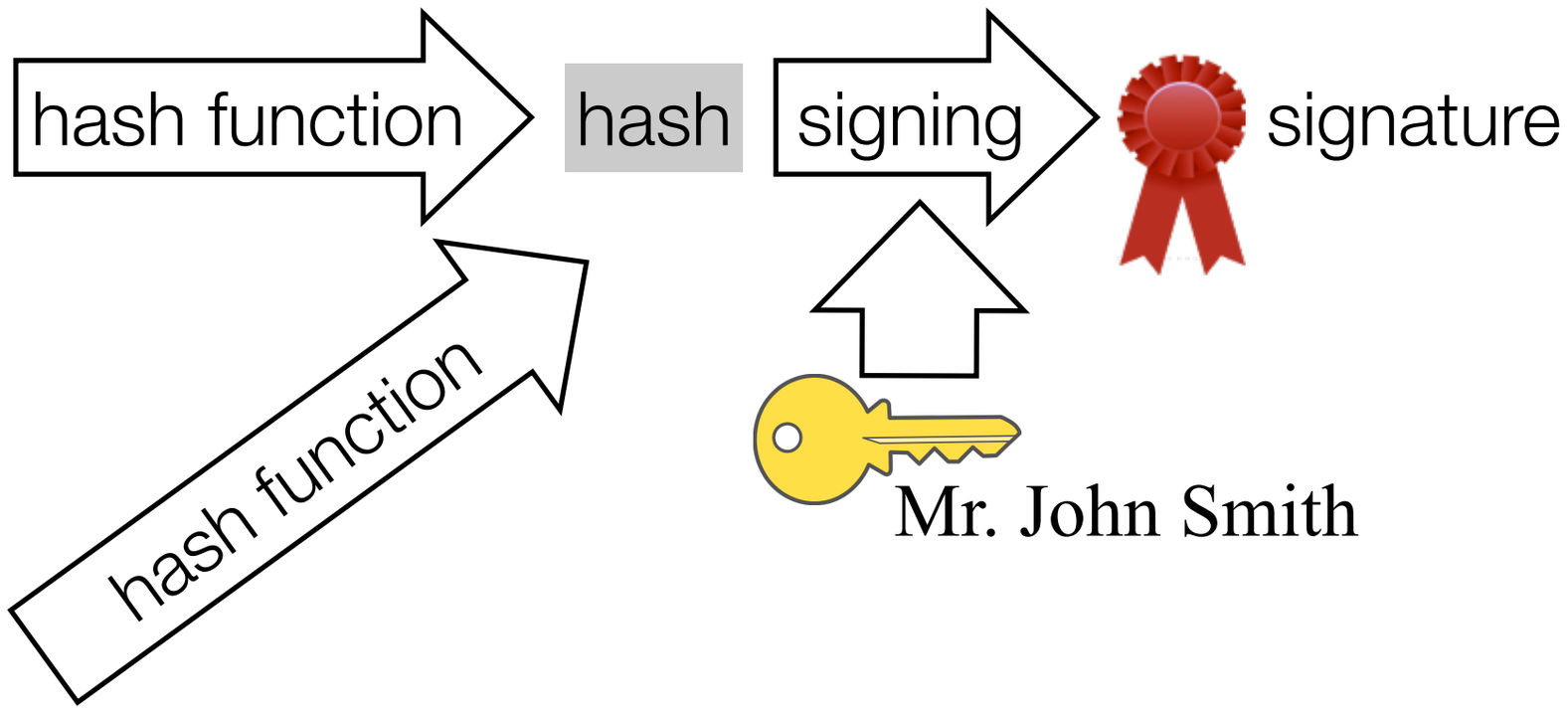
- Suppose that two different inputs have the same hash value:

Contract

Mr. John Smith agrees to sell his yacht to Mr. John Doe in exchange for $2,000,000.

Contract

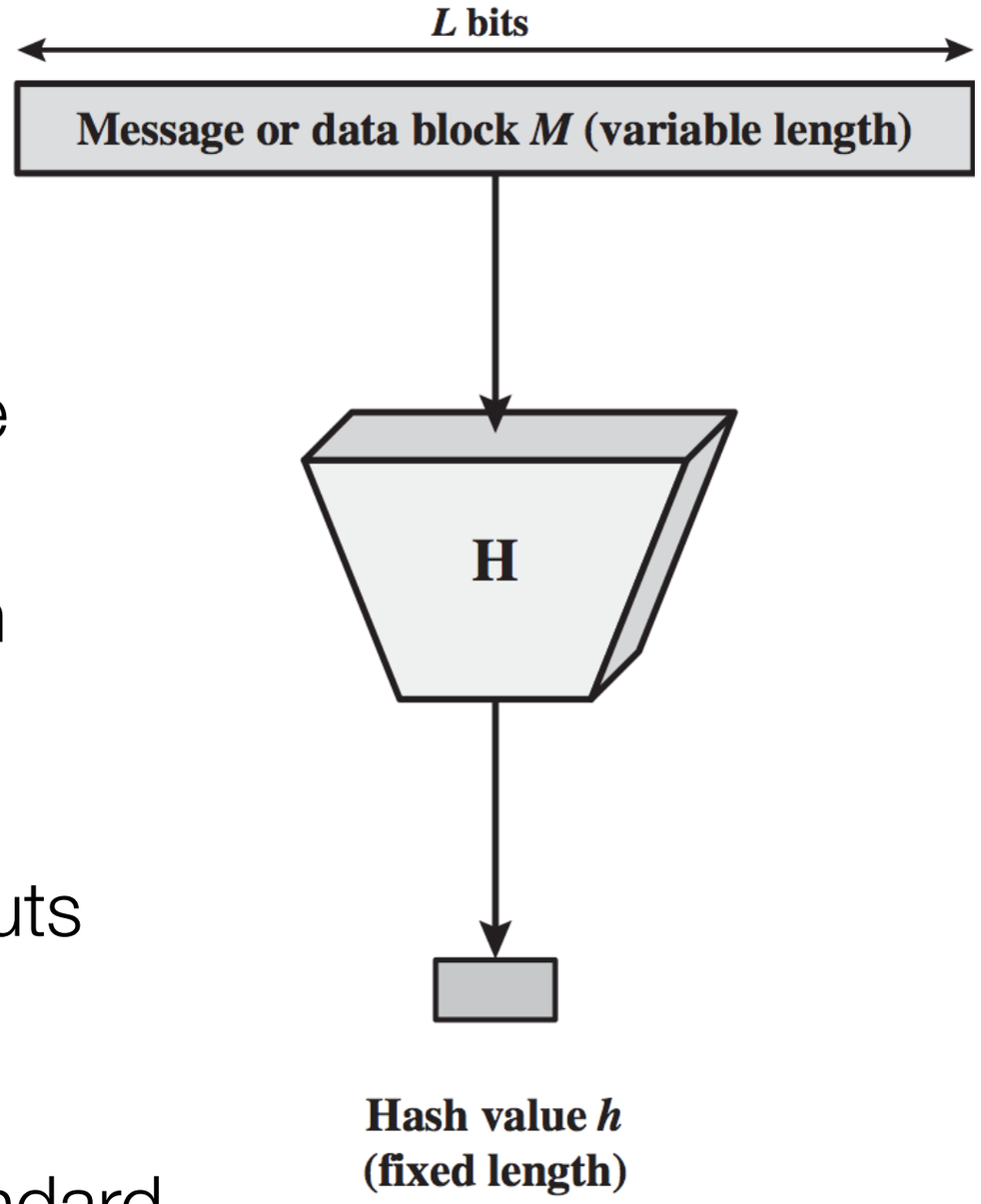
Mr. John Smith agrees to sell his yacht to Mr. John Doe in exchange for $2,000.



- **Collision-resistant hash function**:
  it is hard to find two inputs having the same hash value

Cryptographic Hash Functions

- **Hash function H**:
  deterministically maps an input **M**
  to a fixed-length hash value **H(M)**
- Requirements
  - **efficient**: computing the hash value of a given input is easy
  - **one-way**: finding an input for which the output is a given hash value is hard (i.e., function is hard to invert)
  - **collision-resistant**: finding two inputs for which the hash values are the same is hard
  - **pseudorandom**: output meets standard tests for pseudo-randomness

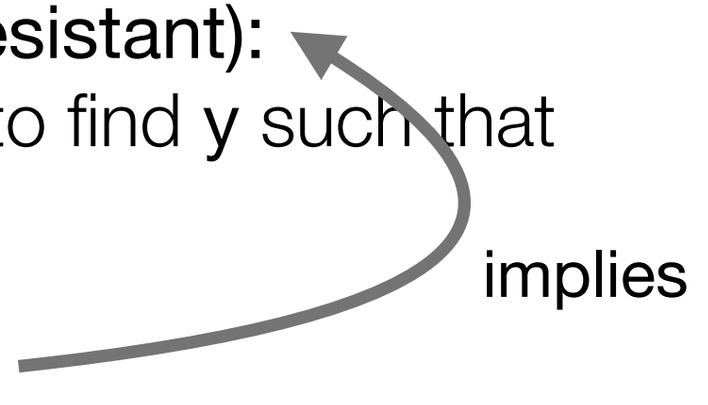


Security Requirements

1. **Preimage resistant (one-way property):**
   given hash value **h**, it is computationally infeasible to find input **y** such that **H(y) = h**

2. **Second preimage resistant (weak collision resistant):**
   given input **x**, it is computationally infeasible to find **y** such that **x ≠ y** but **H(x) = H(y)**

3. **Collision resistant (strong collision resistant):**
   it is computationally infeasible to find any pair of inputs **(x, y)** such that **H(x) = H(y)**



- Collision resistance implies second preimage resistance
  - suppose that there is an attack that finds a second preimage for any input **x**
  - then, choose an arbitrary **x** and use this attack to find an **y** with the same hash
    → there is an attack that finds a collision **(x, y)**

Brute-Force Attacks

Brute-force: try random inputs until a preimage or collision is found

- Preimage (or second preimage) attacks
  - given a hash value (or an input), find an input with the same hash value
  - output is **m** random bits
    - → probability of success for one try is 2^{-m}
    - → on average, attacker needs 2^{m} tries for one successful input
- Collision resistance attacks
  - find two inputs with the same hash value
  - much easier due to the **birthday paradox**

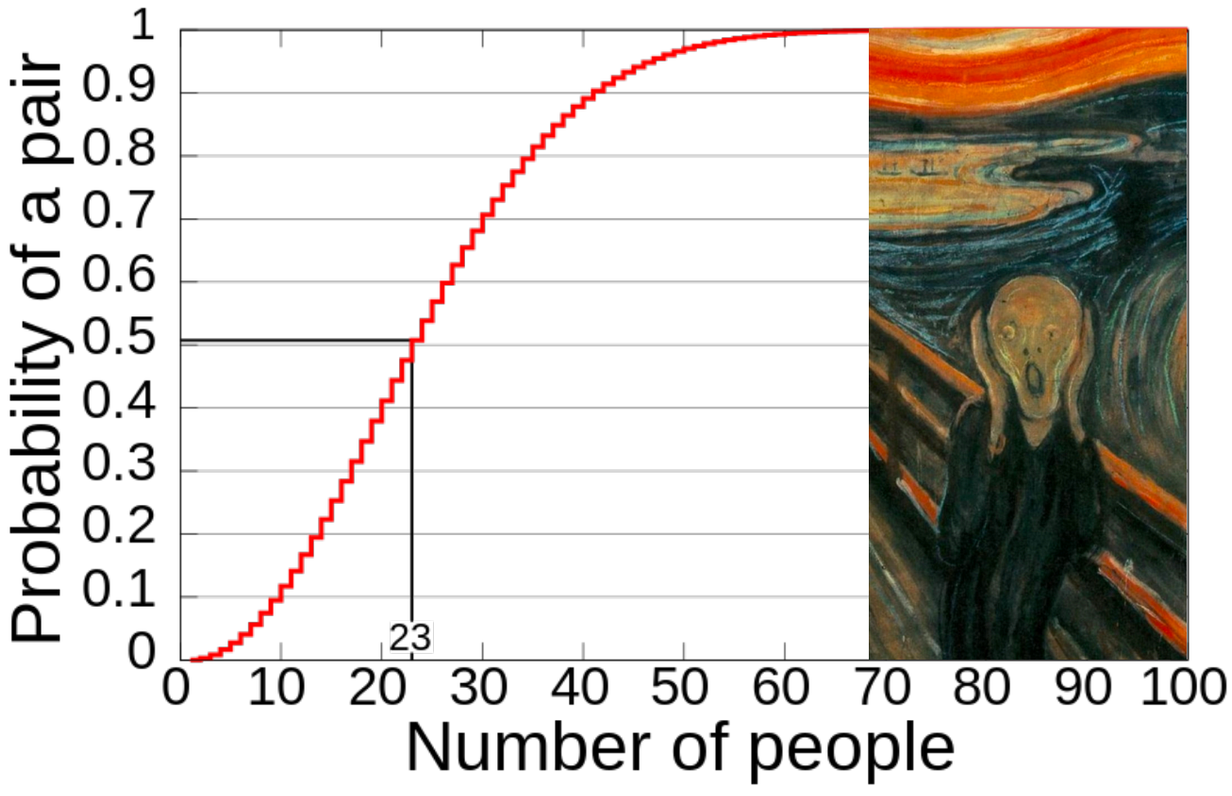
Birthday Paradox

- Probability that at least two people share a birthday in a group of **N** people
  - probability of not sharing = $1 \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-(N-1)}{365}$

$$= \prod_{i=1}^{N-1} \frac{365-i}{365} \approx e^{\frac{-N^2}{730}}$$

  - prob. of sharing reaches **50%** around at $N = 23$

- Generally, if we draw **N** values at random from a set of **M** elements, a collision is likely if $N > \sqrt{M}$



Birthday Attack

- For an **m**-bit hash value, we need around $\sqrt{2^m} = 2^{m/2}$ inputs for a collision
- Generating large number of meaningful inputs

Dear Anthony,

{ This letter is / I am writing } to introduce { you to / to you } { Mr. / -- } Alfred { P. / -- }

Barton, the { new / newly appointed } { chief / senior } jewellery buyer for { our / the }

Northern { European / Europe } { area / division } . He { will take / has taken } over { the / -- }

- Collision between “honest” and “malicious” inputs: two sets, honest and malicious, both of cardinality $\sqrt{2^m} = 2^{m/2}$
- Hash functions must have long outputs
  - *example:* SHA-2 is **224 - 512 bits** ↔ AES is **128 - 256 bits**

Designing Hash Functions

Non-Cryptographic Hash Functions

- Hash functions (or checksum algorithms) for **error detection**
  - much cheaper computationally than cryptographic hash functions
  - may provide error correction as well
  - do **not** provide security

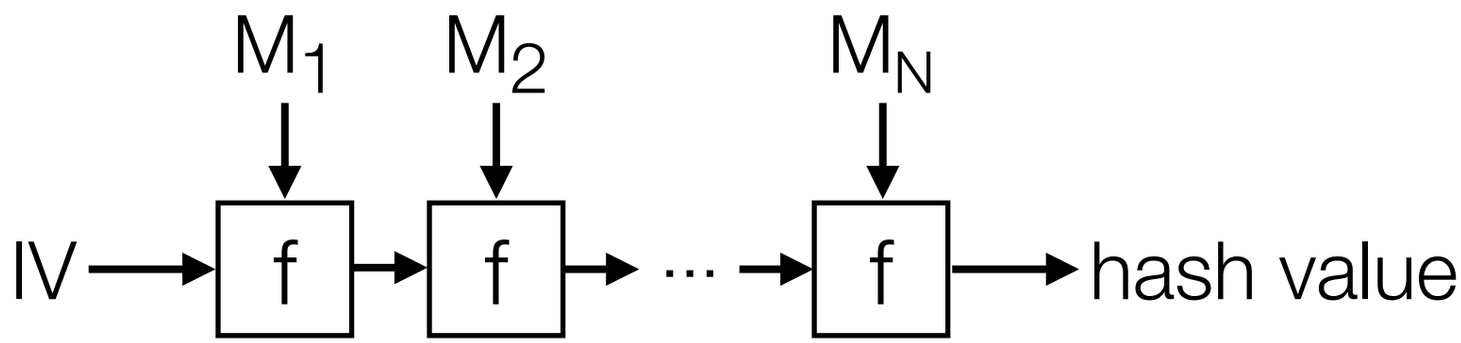
- *Example*:

Cyclic Redundancy Check (CRC)

- very widely used (*e.g.*, cell-phone networks, Ethernet)
- without pre- or post-processing, it is linear with respect to XOR:
  $CRC(x) \oplus CRC(y) = CRC(x \oplus y)$
- very easy to find collisions or input with certain output

Iterative Hash Functions

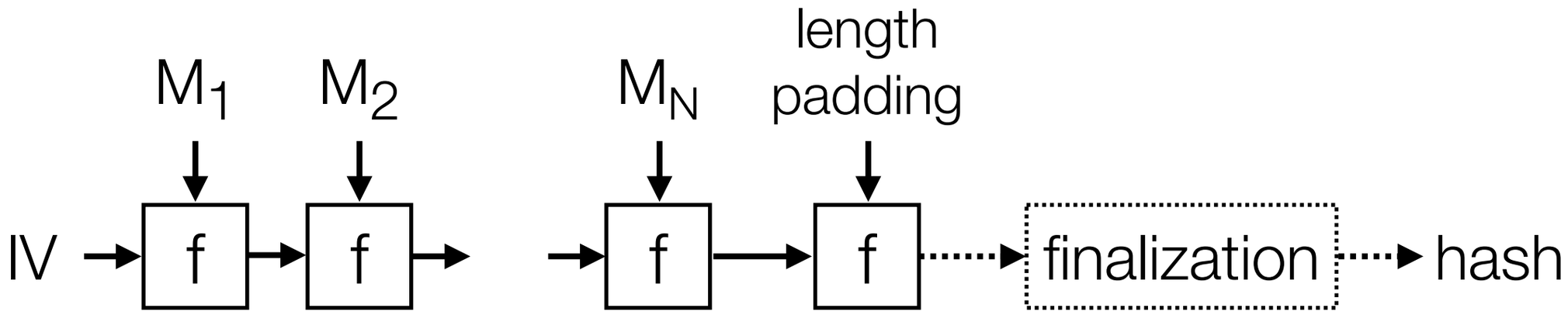
- Divide input M into fixed-length blocks M_1, M_2, …, M_N
- Iterative hash function



- **IV**: initialization vector
- **f**: compression function
  - one-way and collision-resistant
  - takes two fixed-length inputs, produces one fixed-length output

Merkle-Damgård Construction

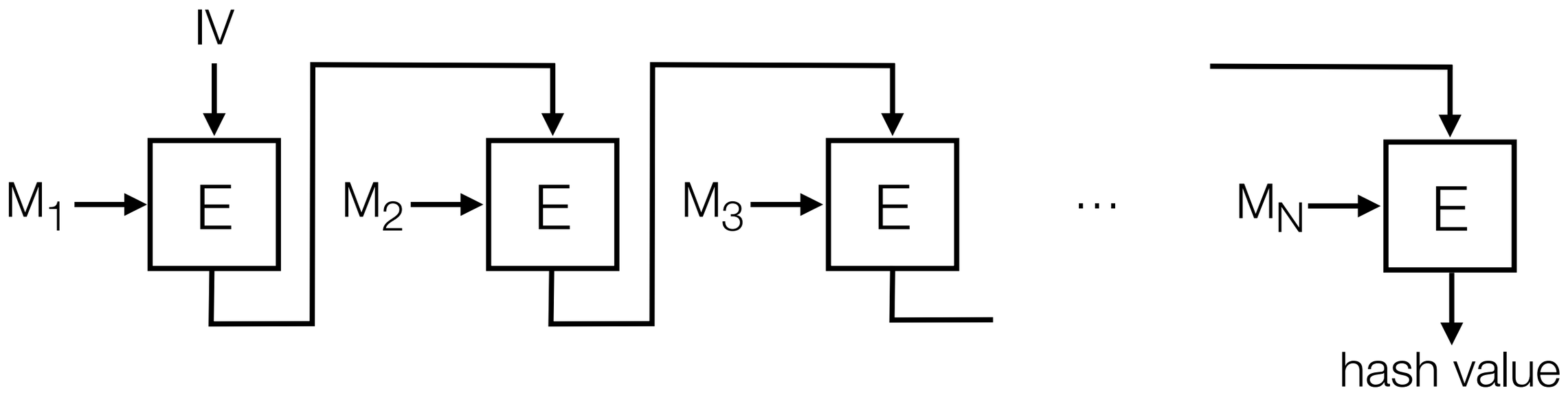
- General method for building cryptographic hash functions from collision-resistant one-way compression functions
  - a lot of widely used hash functions are based on this construction, *e.g.*, MD5, SHA-1, SHA-2



  - **length padding** (Merkle-Damgård strengthening): includes the length of the input as well as a fixed pattern

- Provably secure:
  if the compression function is collision resistant and a proper length padding is added, the resulting hash function is also collision resistant

Hash Functions Based on Cipher Block Chaining



- Similar to the CBC block cipher mode, but there is no secret key
- Output length of typical block ciphers is too short
  - 3DES: **64 bit** → collision attack requires 2^{32} inputs
  - AES: **128 bit** → collision attack requires 2^{64} inputs
  - meet-in-the-middle style attack against (second) preimage resistance has the same complexity as a collision attack

MD5

- Designed by Ronald Rivest in 1991, published in 1992
- Properties
  - based on Merkle-Damgård construction
  - compression function is based on four rounds, each consisting of 16 operations
  - 512-bit block length
  - **128-bit hash length**
    → very short for a hash function (today)
- MD5 was very widely used in practice for various applications (*e.g.*, digital signatures for HTTPS)
- First weakness was found in 1996

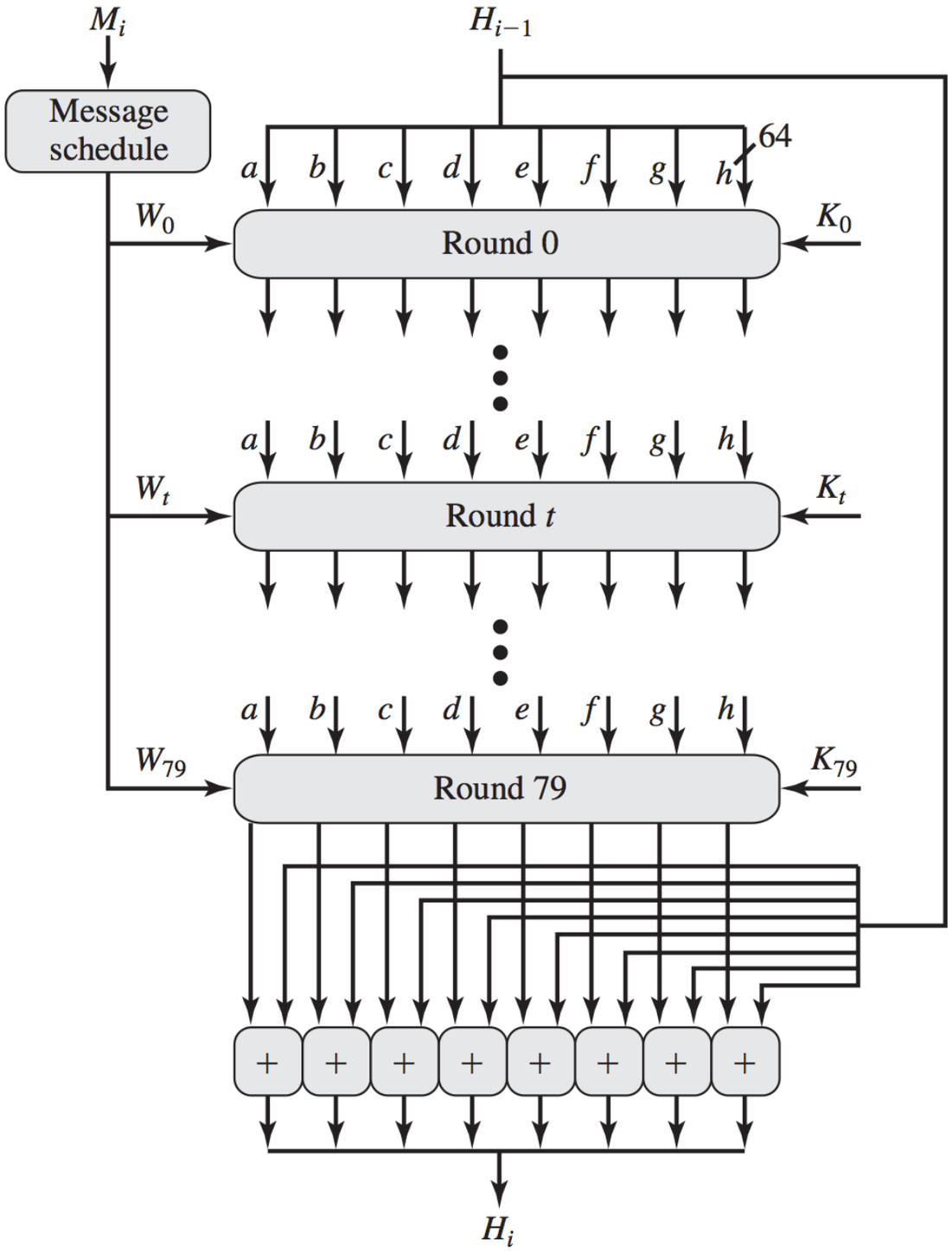
MD5 Vulnerability

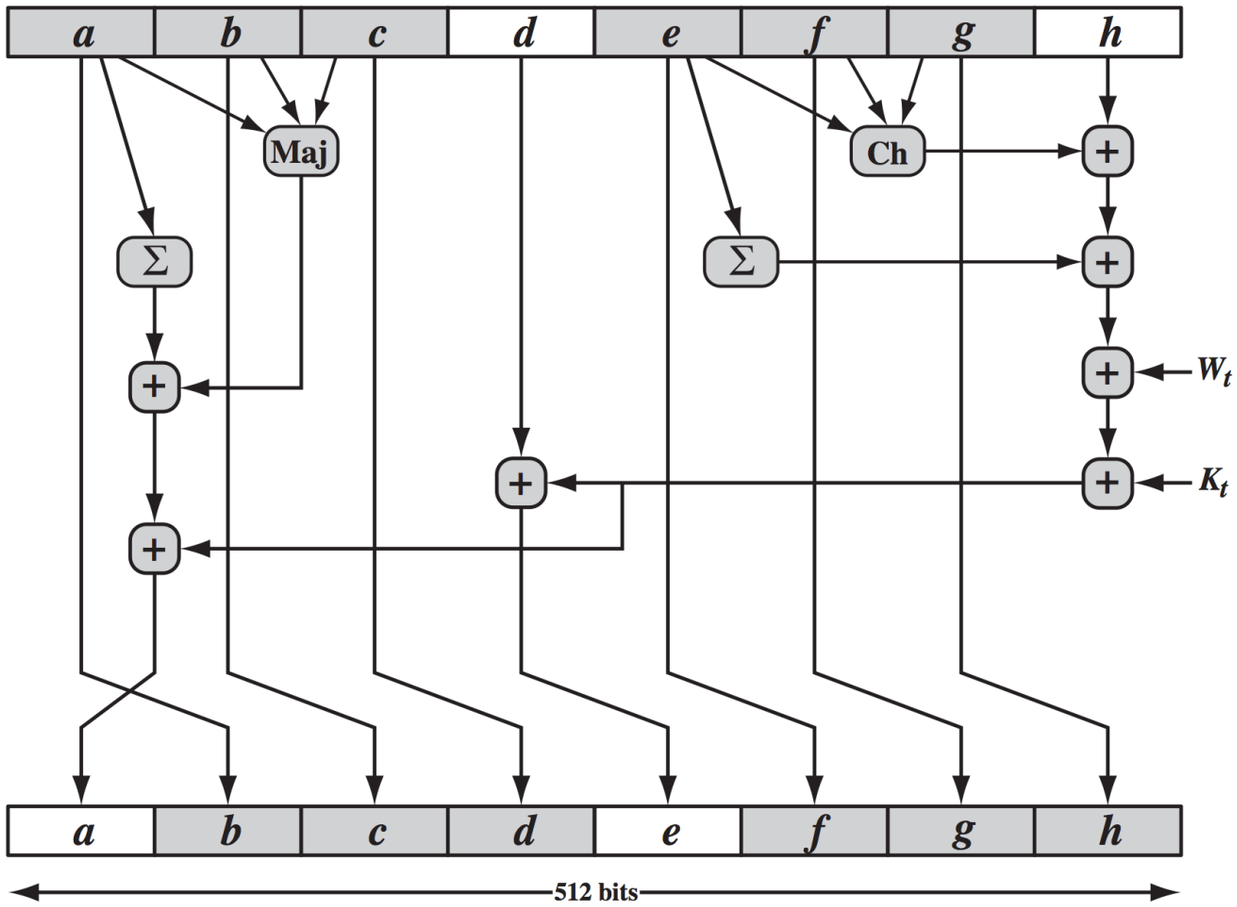
- In 2004, it was demonstrated that MD5 is not collision resistant
- Flame malware (2012)
  - similar to the Stuxnet worm, used for cyber-espionage in the Middle East
  - some components of the malware were **digitally signed with a fraudulent certificate** to make them **appear to originate from Microsoft**
  - certificate was signed using an **MD5 hash value**
- Best public cryptanalysis (2013):
  breaks collision-resistance in 2^{18} steps
  → less than a second on an average computer

Secure Hash Algorithm (SHA)

- SHA-1
  - designed by NSA, published by NIST as FIPS PUB 180-1 in 1995
  - 160-bit hash value, Merkle-Damgård construction
  - it was shown in 2001 that a collision can be found in 2^{65} steps
- SHA-2
  - designed by NSA, published by NIST as FIPS PUB 180-2 in 2002
  - also published as the Internet standard RFC 6234
  - **family of functions: SHA-224, SHA-256, SHA-384, and SHA-512**
    - produce 224, 256, 384, and 512-bit outputs, respectively
  - same structure and underlying operations as SHA-1
  - some weaknesses have been found

SHA-512 Compression Function





Maj, Ch, Σ: Boolean functions defined in the standard

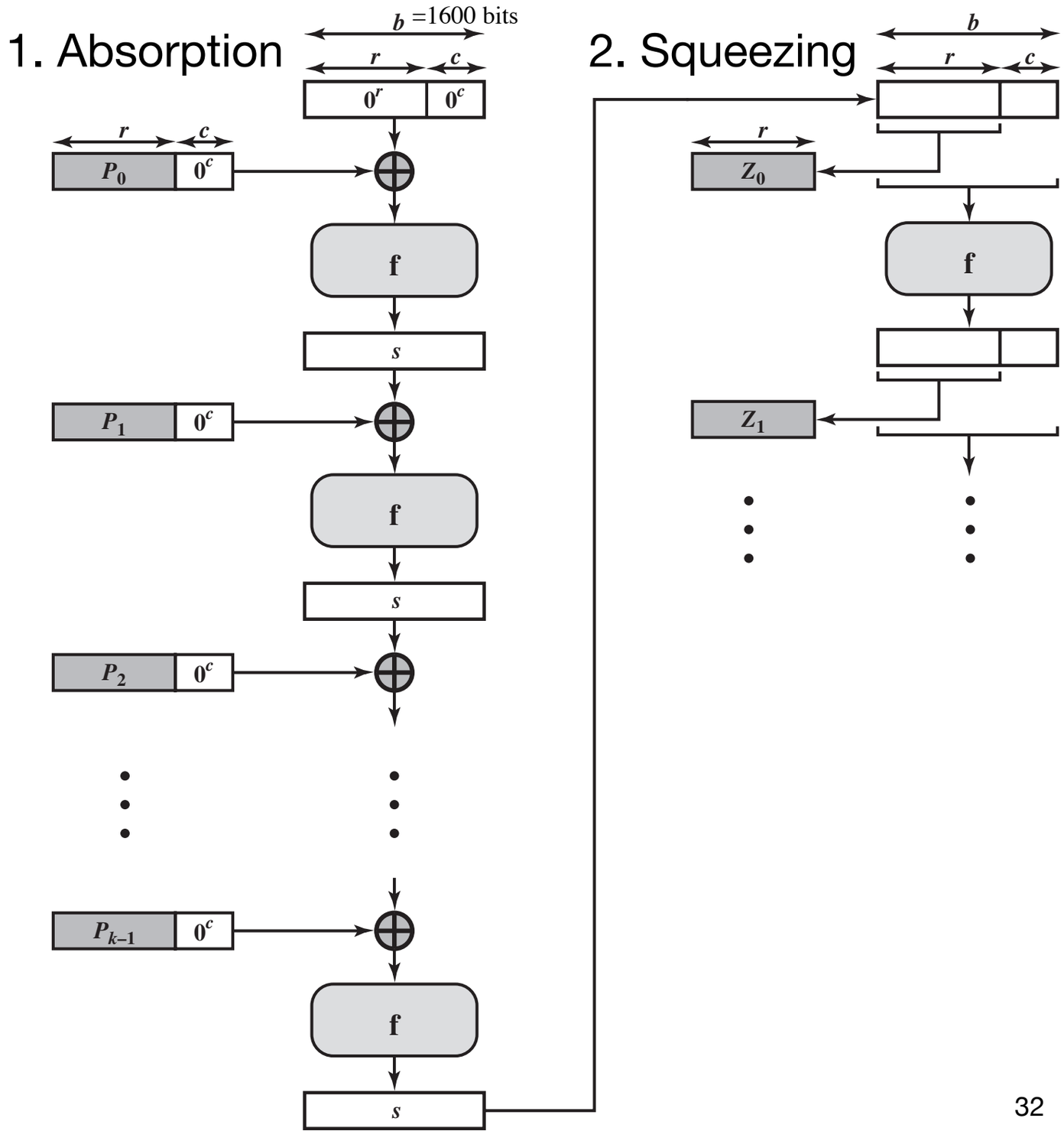
K_t: round constants

SHA-3

- No practical attacks against SHA-2 are known (yet), but a suitable replacement had to be found in time
- In 2007, **NIST announced a competition to develop** a new hash function called **SHA-3**
- In 2012, the Keccak hash function was selected as the winner
  - designed by Bertoni, Daemen, Peeters, and Van Assche
- In 2015, NIST published the SHA-3 standard
- SHA-3 uses the **sponge construction**
- Output length can be **arbitrary**

Sponge Construction

- Data is “absorbed” into the sponge, and the result is “squeezed” out
  - **f**: iteration function
  - **r**: “bitrate” (i.e., how many input bits are absorbed in each iteration)
- **c**: capacity, added for security
- default:
  c = 1024 bits, r = 576 bits
- padding:
  100…01
  (so each block is **r**-bits long)



Iteration Function

- Consists of 24 rounds of processing
- In each round, there are five operations
  - θ (theta) } permutation
  - ρ (rho)
  - π (pi)
  - χ (chi) } substitution
  - ι (iota)
- Operations are based on bitwise boolean operations (XOR, AND, NOT) and rotations
  - can be efficiently implemented in either hardware of software

Conclusion

- Cryptographic hash functions
  - **arbitrary-length input** into **fixed-length output**
  - **one-way** and **collision-resistant**
- In practice,
  - MD5: not secure at all
  - SHA-1: not secure
  - SHA-2: mostly secure
  - SHA-3: secure

Next lecture:

Integrity