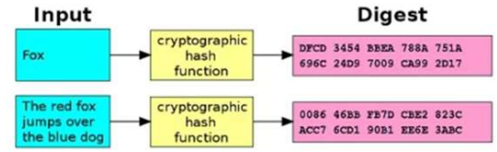
Lecture 7: Hash Functions

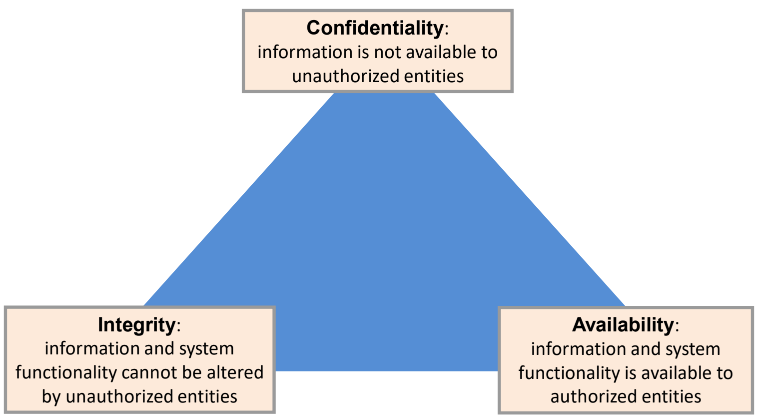
Stephen Huang

Content

1. Integrity
2. Hash Functions
   - Cryptographic Hash Functions
   - Digital Signature
3. Designing Hash Functions

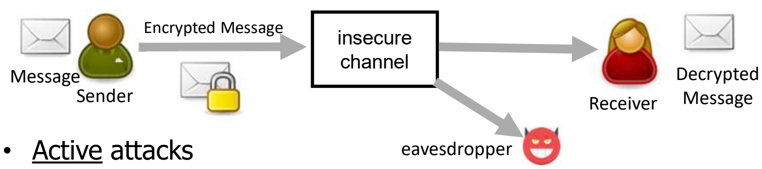


1. Review

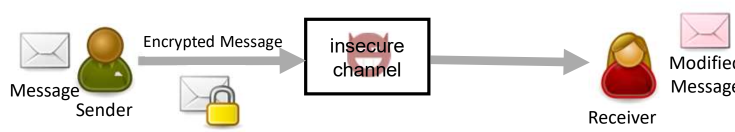


Attacks Against Communication

- Protecting confidentiality against passive attacks.

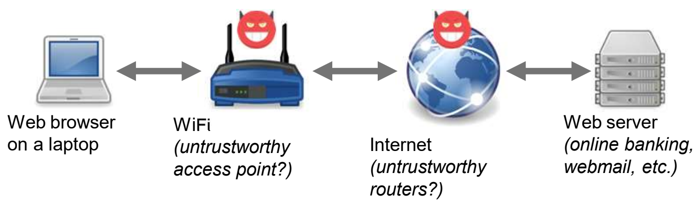


- Active attacks



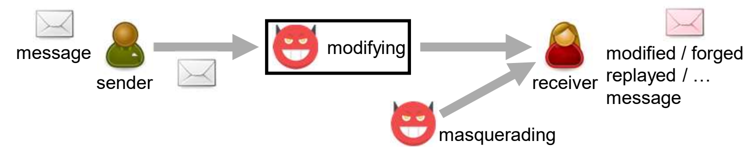
- Data integrity: information cannot be modified in an unauthorized and undetected manner.

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 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.

Tampering with Stream Ciphers

- Stream ciphers: $C = P \oplus G(K)$ and $P = C \oplus G(K)$
  - modified ciphertext: $C' = \Delta \oplus C$
  - modified plaintext: $P' = C' \oplus G(K) = \Delta \oplus C \oplus G(K) = \Delta \oplus P$
- Plaintext

`Transfer one million dollars to Mr. John Smith's account.`

- Ciphertext

`11DE8aAs7gzUovteKIy6G7yttaacP5pFcGPW3m54Nr4Hepd17kAjr4kfs`

⊕ ("Smith's" ⊕ "Doe's ")

- Ciphertext

`11DE8aAs7gzUovteKIy6G7yttaacP5pFcGPW3m54Nypj9xhJ7kAjr4kfs`

- Plaintext

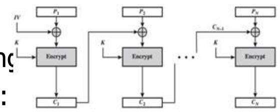
`Transfer one million dollars to Mr. John Doe's   account.`

Tampering with Stream Ciphers

Original Plaintext:		Y	E	S
Binary Representation:		01011010	01000101	01010011
A Pseudorandom Seq.:	⊕	11010010	00100000	11110101
Original Ciphertext:		10001011	01100101	10100110
Modified Ciphertext:		10011100	01101111	11010100
Pseudorandom Seq.:	⊕	11010010	00100000	11110101
Binary Representation:		01001110	01001111	00100001
Modified Plaintext:		N	O	!

Tampering with Block Ciphers

- Single block in Electronic Code Book (ECB) mode: secure against tampering
- Rearranging blocks (e.g., CBC mode):



Cipher Block Chaining

Plaintext

https://www.e	xample.com/i	index.html?pa	ssword=secret

Ciphertext

dgyACJVKcERNl	z9iIcfkeBEYE2	spluELybLi3wm	fq6aSDNIa6wn6

Modified ciphertext

dgyACJVKcERNl	spluELybLi3wm	fq6aSDNIa6wn6	dgyACJVKcERNl	z9iIcfkeBEYE2	spluELybLi3wm

Modified plaintext

https://www.e	wFR0yVk1UrIx0	ssword=secret	5YRnb75iDRSFx	xample.com/i	index.html?pa

Tampering with Block Ciphers

- Original plaintext

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

- Original ciphertext

dgyACJVKcERNl	z9iIcfkeBEYE2	spluELybLi3wm	fq6aSDNIa6wn6	5YRnb75iDRSFx	wFR0yVk1UrIx0

- Modified ciphertext

dgyACJVKcERNl	z9iIcfkeBEYE2	5YRnb75iDRSFx	fq6aSDNIa6wn6	spluELybLi3wm	wFR0yVk1UrIx0

- Modified plaintext

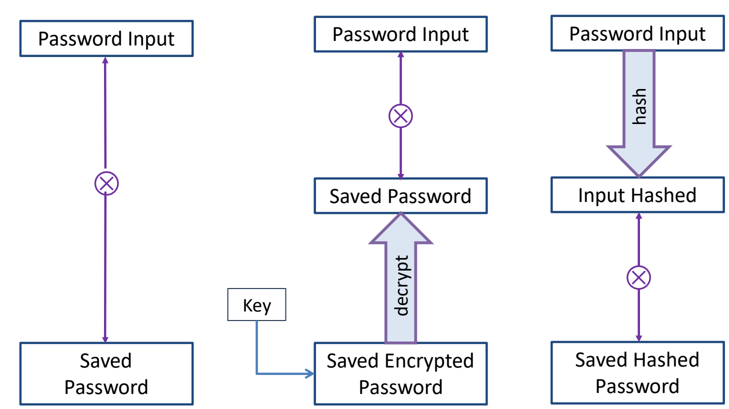
Transfer one	million USD to	John Doe's	account from	John Smith's	account.

2. Hashing

Hash Function Applications

- Used Alone
  - Fingerprint: file integrity verification
  - Password storage (one-way encryption)
- Combined with Encryption
  - Message Authentication Code)MAC):
    - protects both a message's integrity and authentication
  - Digital Signature
    - Ensure non-repudiation
    - Encrypt hash with private (signing) key and verify with public (verification) key

2. Hashing

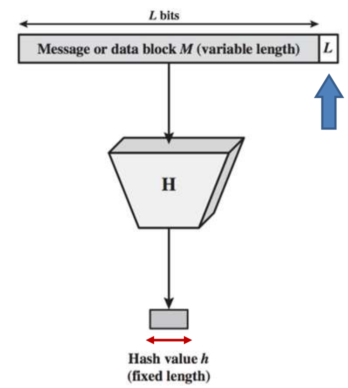


Hash Functions vs. Crypto Primitives

	Input:	Pseudorandom output:
Stream ciphers (i.e., pseudo-random number generators)	Fixed-length Key	Arbitrary-length sequence
Block ciphers, public-key ciphers	Fixed-length block	Fixed-length block
Hash functions	Arbitrary-length input	Fixed-length hash

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., the function is hard to invert).



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? Bad Idea.
  - The secret key would also have to be stored on the server (or be accessible to the server), so hackers could easily decrypt the passwords.

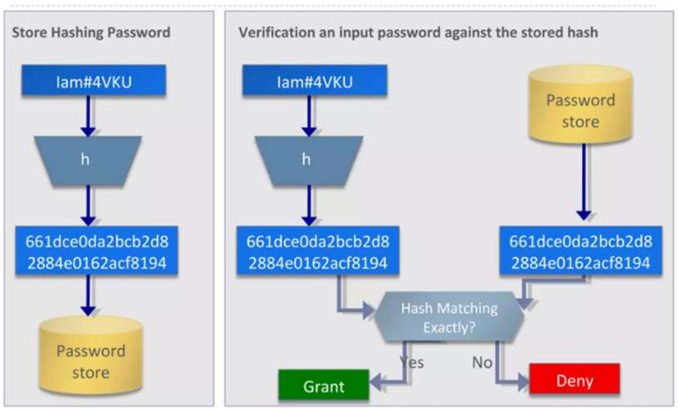
Application: Password File

- 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...
```

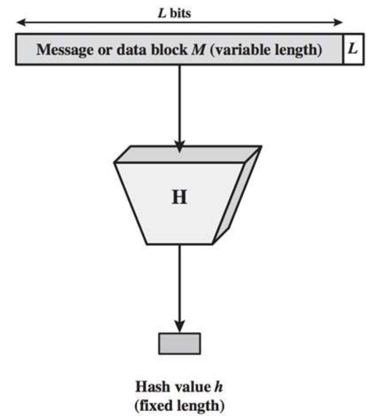
  - if a hacker compromises the server, it will learn only the hash values but not the actual passwords.
- One-way hash function: It is hard to find a password whose hash is a given value.

Password Verification



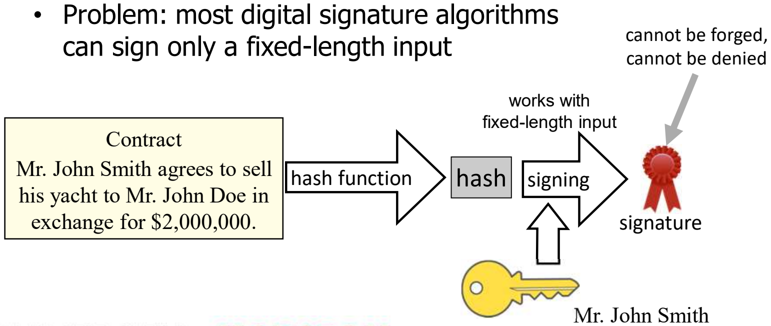
Cryptographic Hash Functions

- 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., the function is hard to invert)
  - **Collision-resistant**: finding two inputs for which the hash values are the same is hard but not impossible.



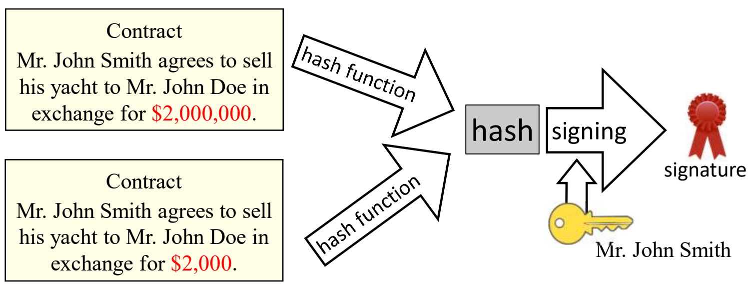
Application: Digital Signatures

- 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



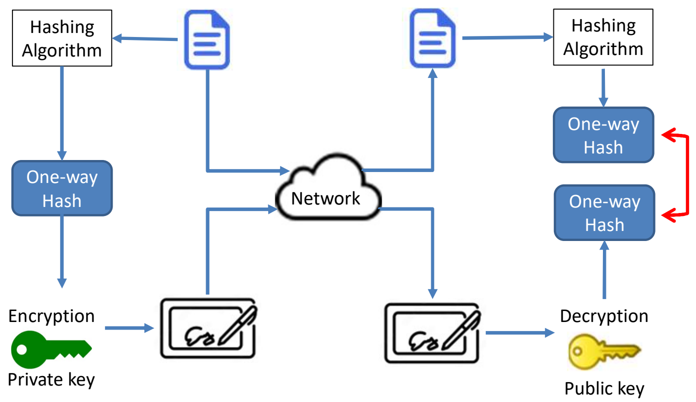
Digital Signature

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



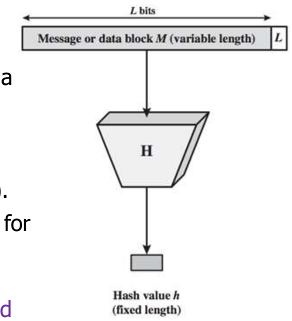
- Collision-resistant hash function: it is hard to find two inputs having the same hash value.

Digital Signature in Action



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., the 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 $x \neq y$ 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 a 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

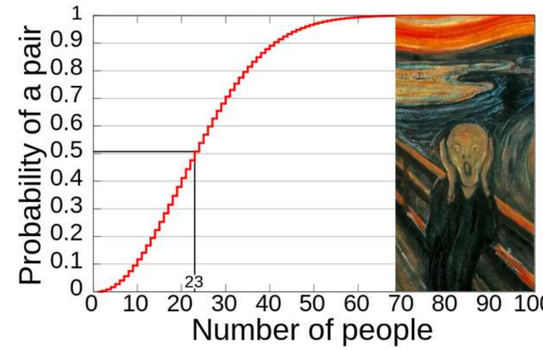
Birthday Paradox

- Probability that at least two people share a birthday in a group of N people
  - probability of not sharing =

$$1 \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ \dots \cdot \frac{365-(N-1)}{365} = \prod_{i=1}^{N-1} \frac{365-i}{365} \approx e^{\frac{-N^2}{730}}$$

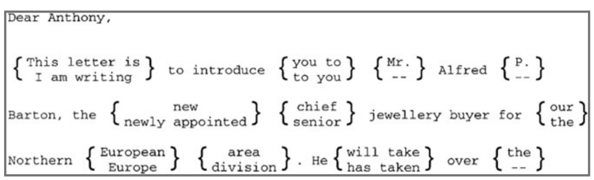
  - prob. of sharing reaches 50% around at $N = \boxed{}$
- Generally, if we draw N values at random from a set of M elements, a collision is likely if $N > \sqrt{M}$.

Birthday Paradox



Birthday Attack

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


Dear Anthony,

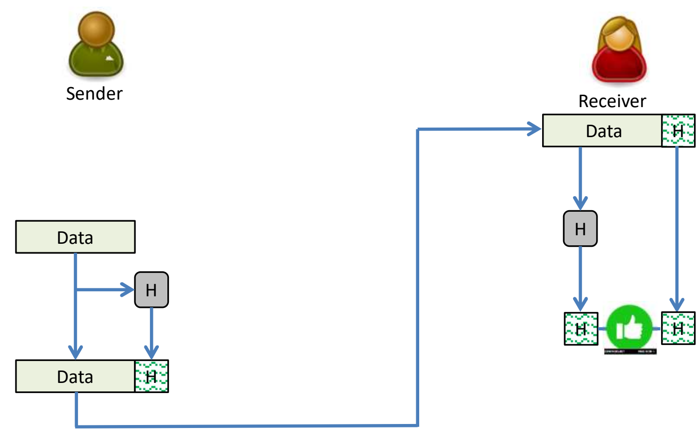
{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}

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

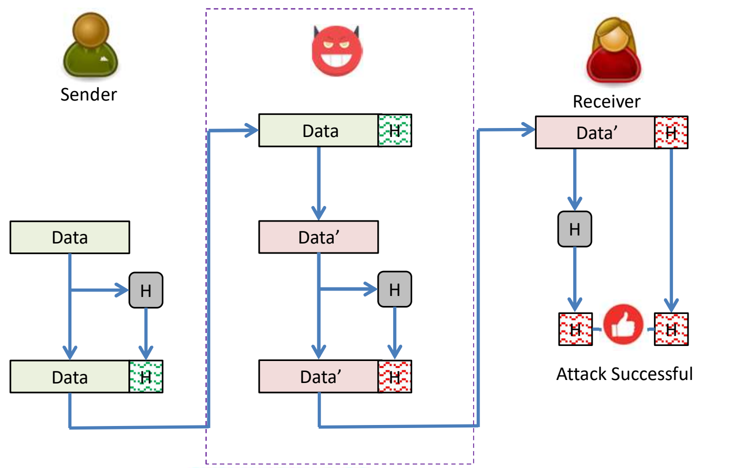
Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --}

- 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 compare to 128-256 bits for AES

Using Hash Functions



Attacks Against Hash Functions



3. Designing Hash Functions

- Non-Cryptographic Hash Functions
- Iterative Hash Functions
- Merkle-Damgård Construction
- Hash Functions Based on Cipher Block Chaining
- MD5
- Secure Hash Algorithms (SHA)

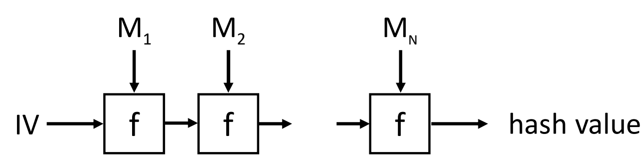
Non-Cryptographic Hash Functions

- 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).
  - It is very easy to find collisions or input with certain output.

Iterative Hash Functions

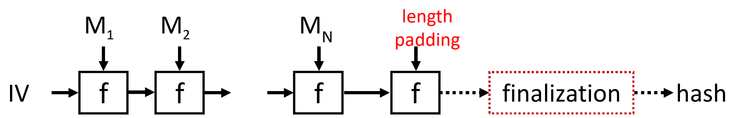
- Divide input M into fixed-length blocks M_1, M_2, …, M_N
- Iterative hash function



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

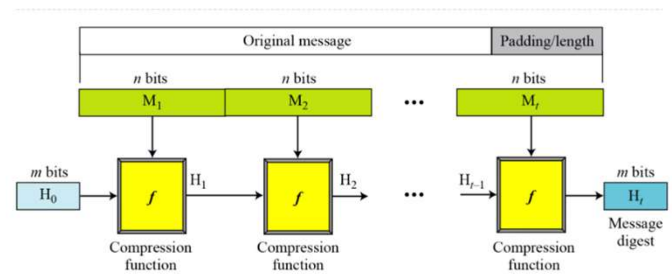
Merkle-Damgård Construction

- 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

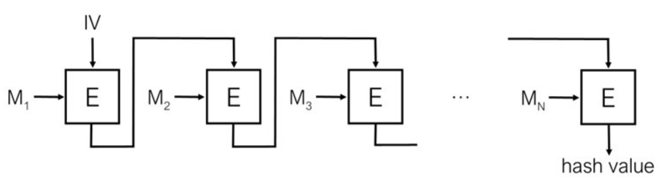


  - length padding (Merkle-Damgård strengthening): includes the length of the input as well as a fixed pattern
- Provably secure: if the compression function is collision resistant and a proper length padding is added, the resulting hash function is also collision resistant.

Merkle-Damgård Construction



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

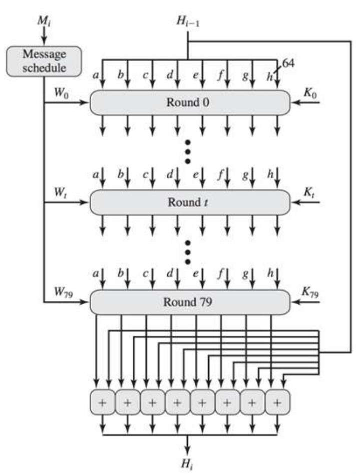
MD5 Vulnerability

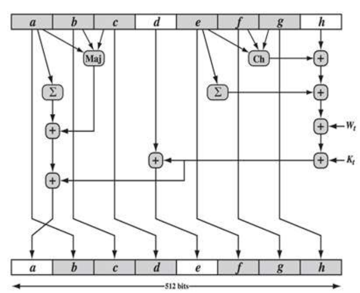
- In 2004, it was demonstrated that MD5 is not collision-resistant
- Flame malware (2012)
  - Similar to the Stuxnet worm, used for cyber-espionage in the Middle East
  - Some components of the malware were digitally signed with a fraudulent certificate to make them appear to originate from Microsoft
  - 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 265 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

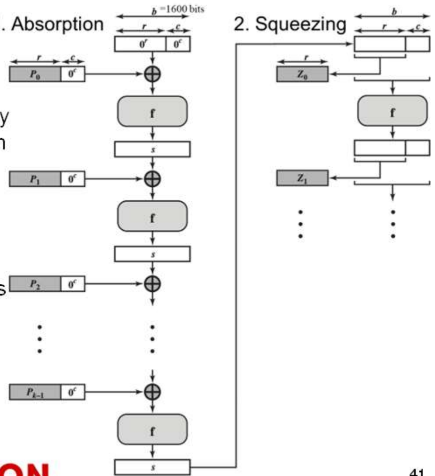
There is no need to memorize this algorithm

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)



Iterative Function

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

- Hash Functions
- **Integrity**
- Key Distribution