Cryptographic Hash Functions

Vijay Ganesh
University of Waterloo

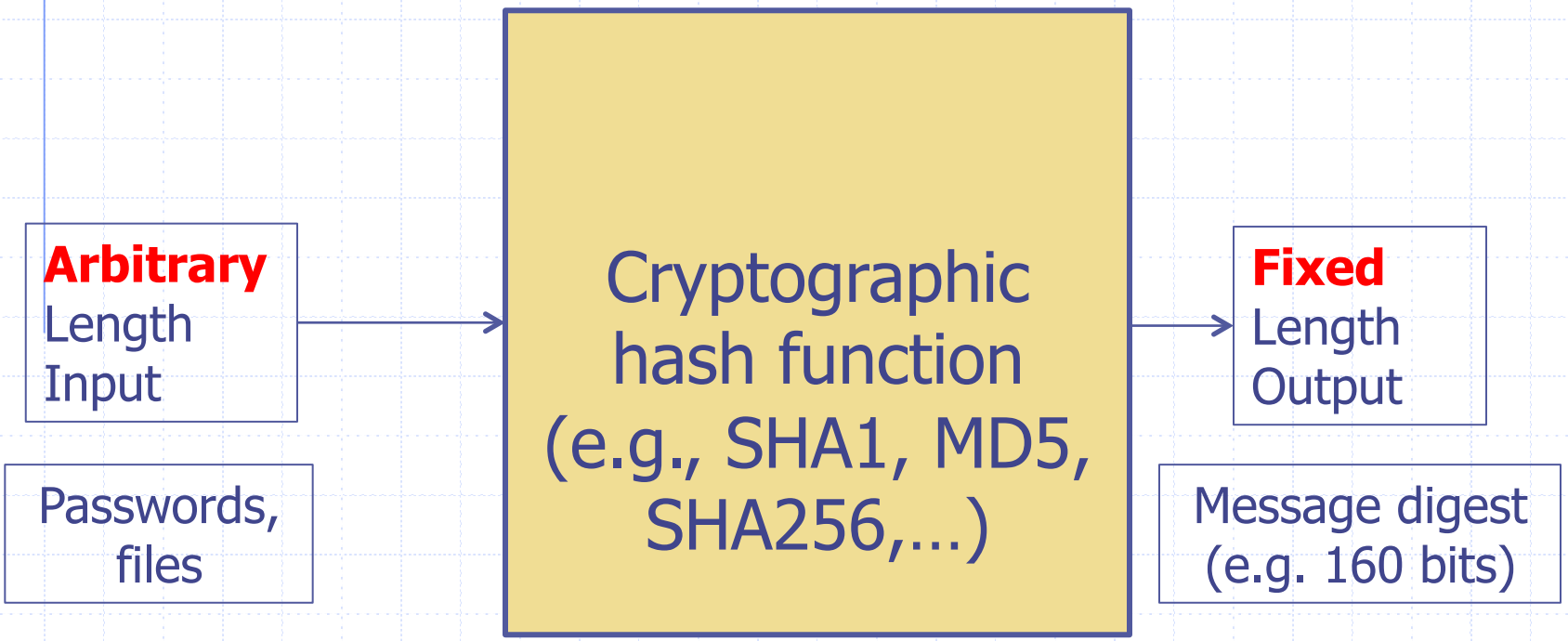
Previous Lecture: Public-key Cryptography

- Motivations for public-key cryptography
- Diffie-Hellman key exchange protocol
- RSA public key encryption scheme
- RSA digital signature scheme

Today's Lecture: Cryptographic Hash Functions

- Basic definition of a hash function (MD2, MD4, MD5, SHA1, SHA256,…)
- Important properties of hash functions
  - E.g., strong collision resistance
- Uses of hash functions
- How hash functions like SHA1 etc. work
  - E.g., Merkle-Damagard construction

Basic Definitions: Cryptographic Hash Functions



- Different from classic hash functions used in hash tables etc.
- Different from "provably-secure" cryptographic hash functions

Uses of Cryptographic Hash Functions

- User Authentication (e.g., passwords)
- Message Authenticity (e.g., Hash MACs)
- Compact file identifiers (e.g., in Git,…)
- Used in digital signatures
- Certain obfuscation schemes rely on “provably-secure” hash functions

Important Properties of Cryptographic Hash Functions

- Compression
- First pre-image resistance
- Second pre-image resistance
- Strong collision resistance
- Efficient
- Deterministic (?)

Important Properties of Cryptographic Hash Functions

- Let h: X ➔ Y denote a hash function
- First pre-image resistance:
  - Given y in Y, it is "computationally infeasible" to compute a value x in X such that h(x) = y
- Hard to invert
- Why we need this property?
  - If hash function is invertible, then it is not useful for crypto applications
  - E.g., password authentication will be broken

Important Properties of Cryptographic Hash Functions

- Let h: X ➔ Y denote a hash function
- Second pre-image resistance:
  - Given x in X, it is "computationally infeasible" to compute a different value x′ in X such that h(x) = h(x′)
- Weak collision-resistance (sometimes also called target collision-resistance)
- Why we need this property?
  - If hash function is not weak collision-resistant, then it is not useful for crypto applications
  - E.g., password authentication will be broken

Important Properties of Cryptographic Hash Functions

- Let h: X ➔ Y denote a hash function
- Strong collision-resistance:
  - It is "computationally infeasible" to find distinct inputs x, x' such that h(x) = h(x')
- Why we need this property?
  - Usability of hash functions
  - Security of hash-then-sign schemes
    - Given h, attacker computes m,m' such that h(m) = h(m')
    - Attacker gives m to Alice to hash-then-sign
    - Alice produces <m, Sign(h(m))>
    - Attacker replaces it with <m', Sign(h(m))>, and claims Alice signed it

Relationship between Properties of Cryptographic Hash Functions

- Strong collision resistance implies weak collision resistance in the Random Oracle Model (ROM)
  - Proof by contradiction
  - Assume h is strongly collision-resistant but not weakly collision-resistant
  - What does it mean to be not weakly collision-resistant: Given a value m we can "quickly" find a different value m' such that h(m) = h(m')
  - Present <m,m'> as a counter-example for the strong collision-resistant claim. QED.
- Similarly, show that strong collision resistance implies first pre-image attack resistance

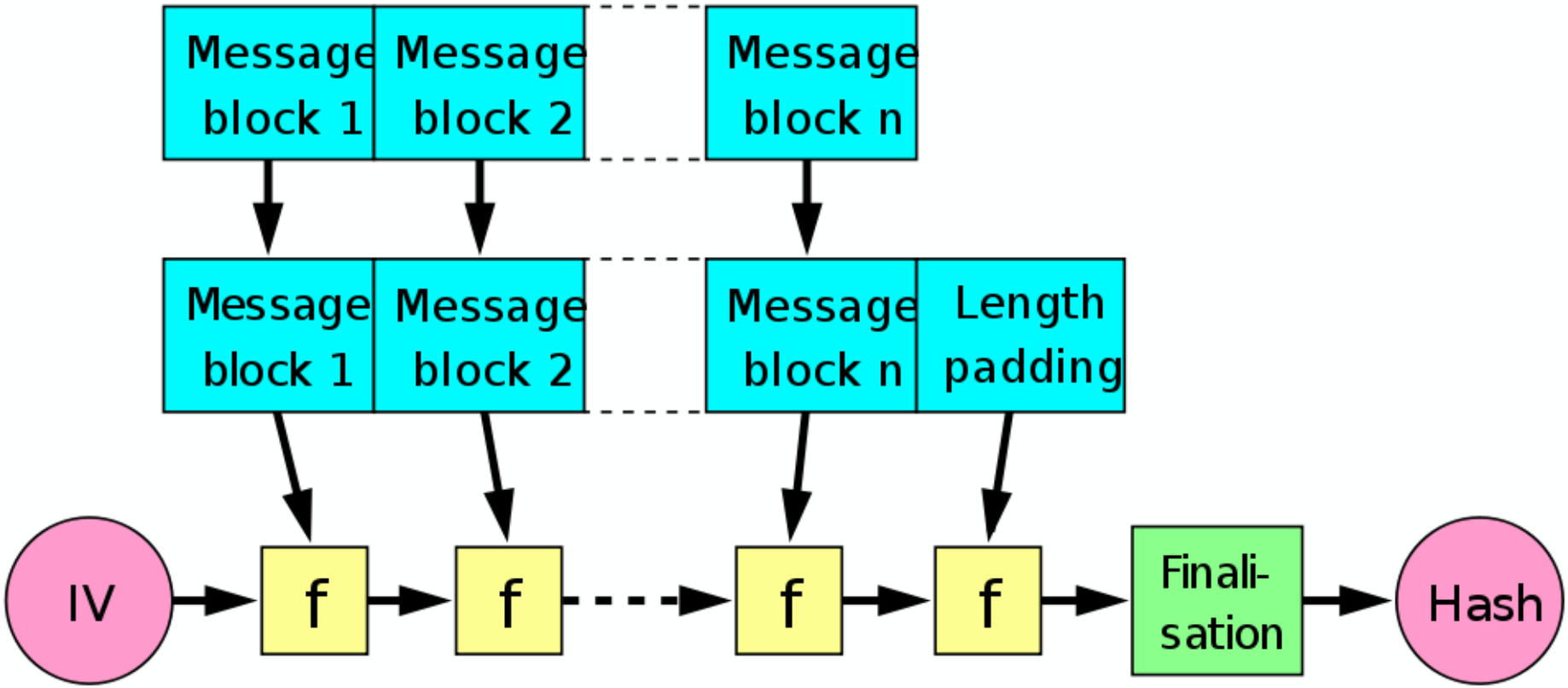
Birthday Paradox and the Cardinality of Hash Function's Range

- Birthday Paradox
  - When iteratively sampling (with replacement) elements from a set of cardinality N, it is highly likely to sample the same element twice after sqrt(N) attempts
- Cardinality of Hash function's range = $2^{(\#\text{ of bits of digest})}$
- Hence, hash function digest size should be as large as possible

How do common hash functions work?

- Most common hash functions such as MD5,… etc. use what is called a Merkle-Damagard (MD) construction
- It is an iterative algorithm, where each iteration is called a round
- Takes an input of arbitrary length, and pads it so that it can be split into equal-size blocks (e.g., 512 bits)
- MD internally uses a proper one-way compression function (Davies-Meyer, MDC-2/Meyer–Schilling,…)

Internals of Hash Functions: Merkle-Damagard Construction



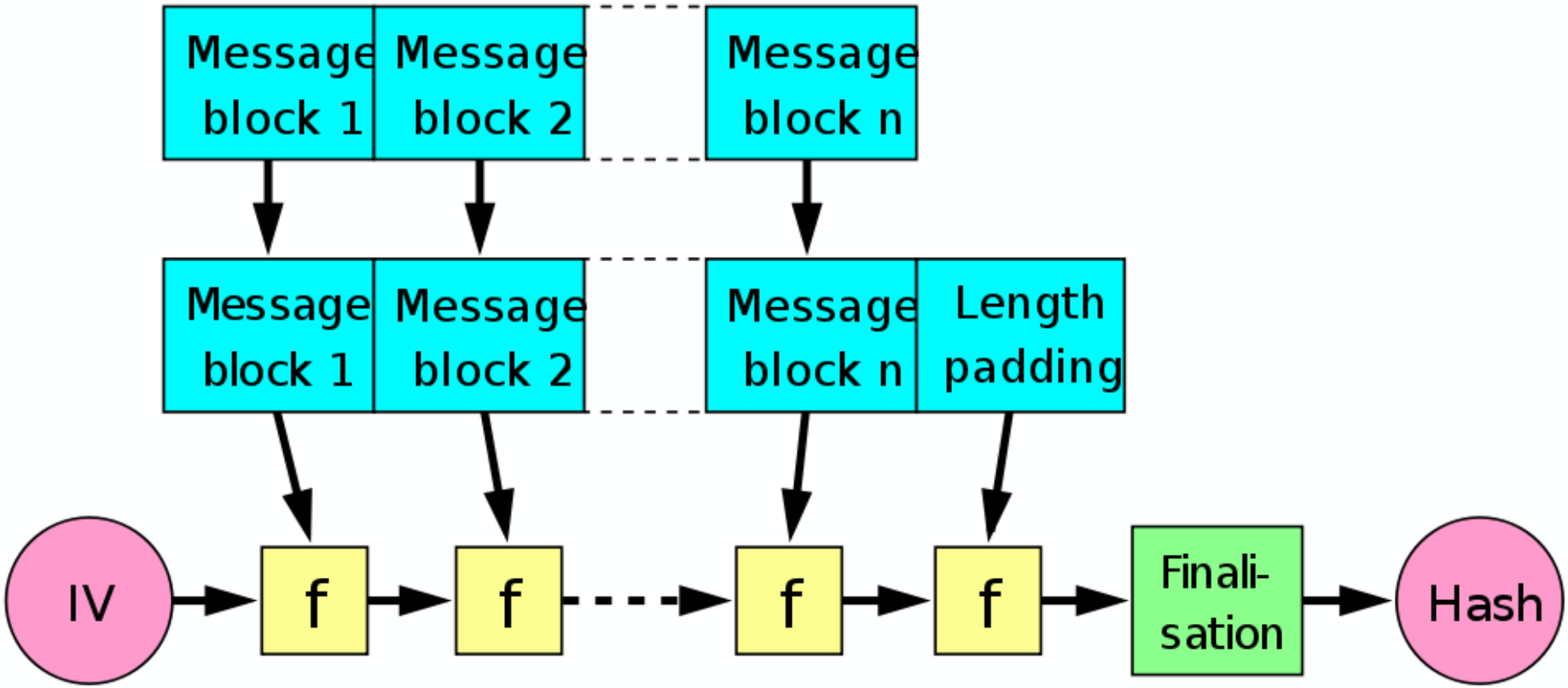
Initialization Vector

Source: Wikipedia

Internals of the Merkle-Damagard Construction

- Questions
  - What is f?
    - f is a compression function whose input and output both are fixed-length strings
  - Does the construction have the various properties we require of hash functions?
  - Intuitively, why is it collision-resistant?
  - Proofs of security?

Internals of Hash Functions: Merkle-Damagard Construction



Initialization Vector

Source: Wikipedia

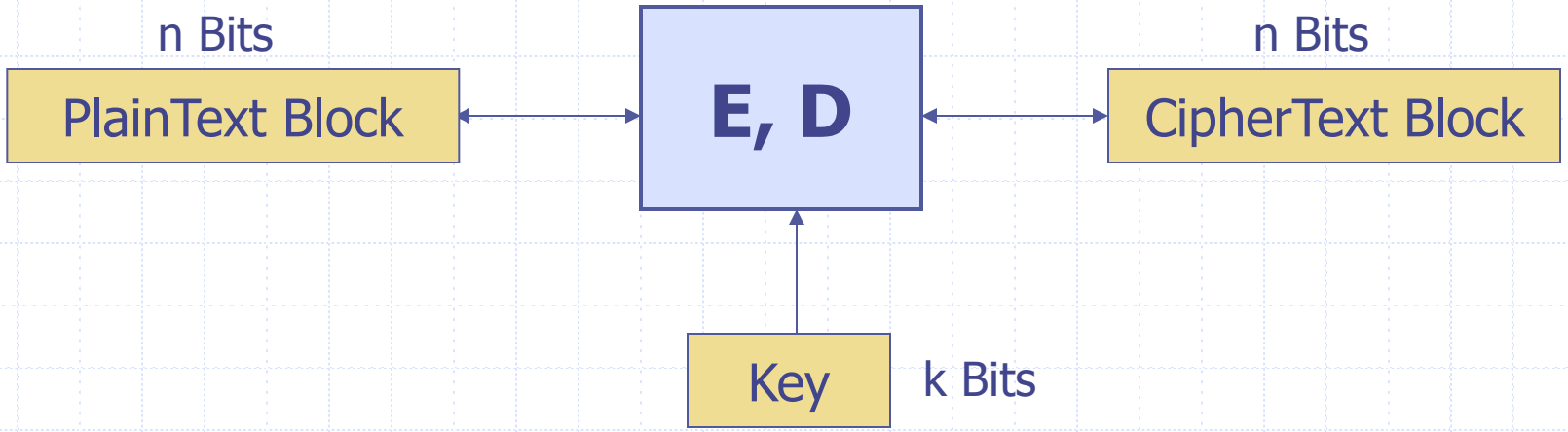
Security of the Merkle-Damagard Construction

- Theorem: If f is a collision-resistant compression function, then h is a collision-resistant hash function
- Proof Sketch: Suppose you can find $x \neq x'$ such that $h(x) = h(x')$, then we can show that we can find collision on f
  - What are the different ways in which we get $h(x) = h(x')$?
    - The last call of f produces the same output for different inputs. We found a collision in f, violating the assumption that f is collision-resistant
    - The last call of f for x, x' have the same input. This means that some previous call of f produces the same output for different inputs, given that x and x' are different at least in 1 bit
    - Hence, we found a collision in f contradicting the assumption

Avalanche Effect: Merkle-Damagard (MD) Construction

- Changing 1 bit of the input can change large number of (upwards of 50%) of the output bits
- A design heuristic for hash functions and block ciphers
- Ensures that similar looking inputs do not produce similar looking outputs
- Essential for security provided by hash-based login
- Difficult to define theoretically and prove

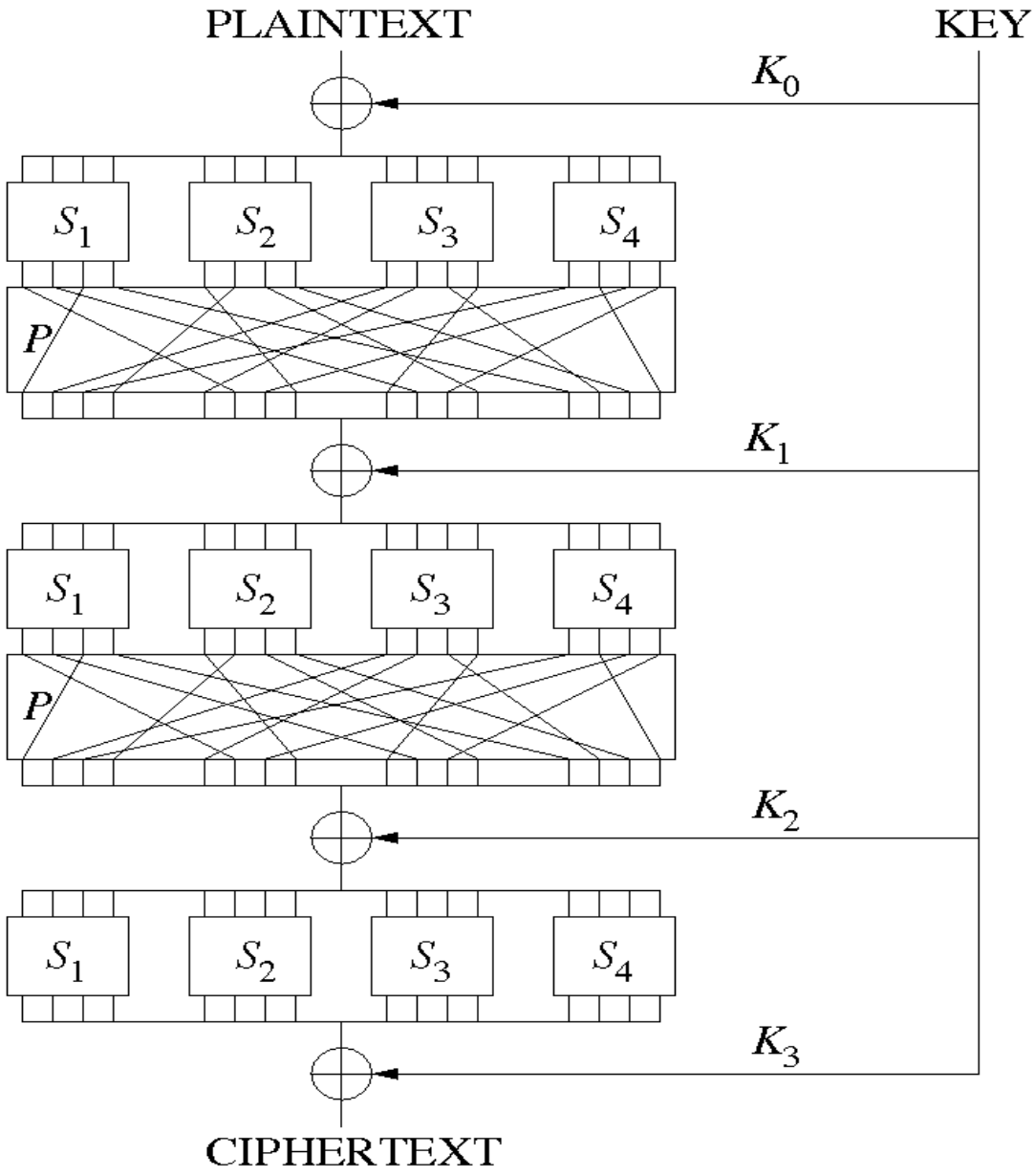
The function ‘f’ built using Block ciphers



Canonical examples:

1. 3DES: n= 64 bits, k = 168 bits
2. AES: n=128 bits, k = 128, 192, 256 bits

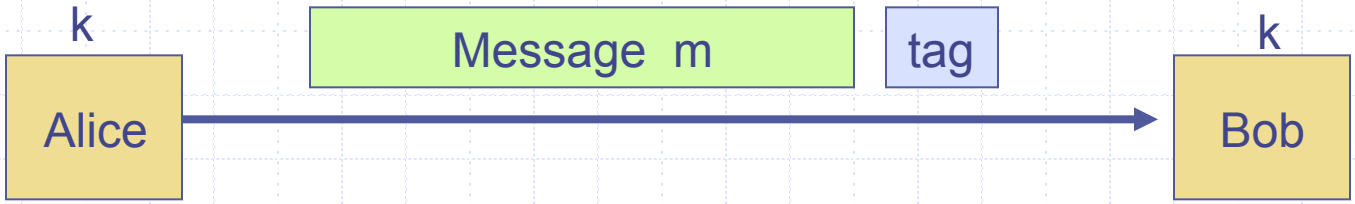
Substitution Permutation Network



Source: Wikipedia

Recalling MACs: Hash Length Extension Attacks

- Goal: message integrity and Authenticity.
- No confidentiality.
  - ex: Protecting public binaries on disk.

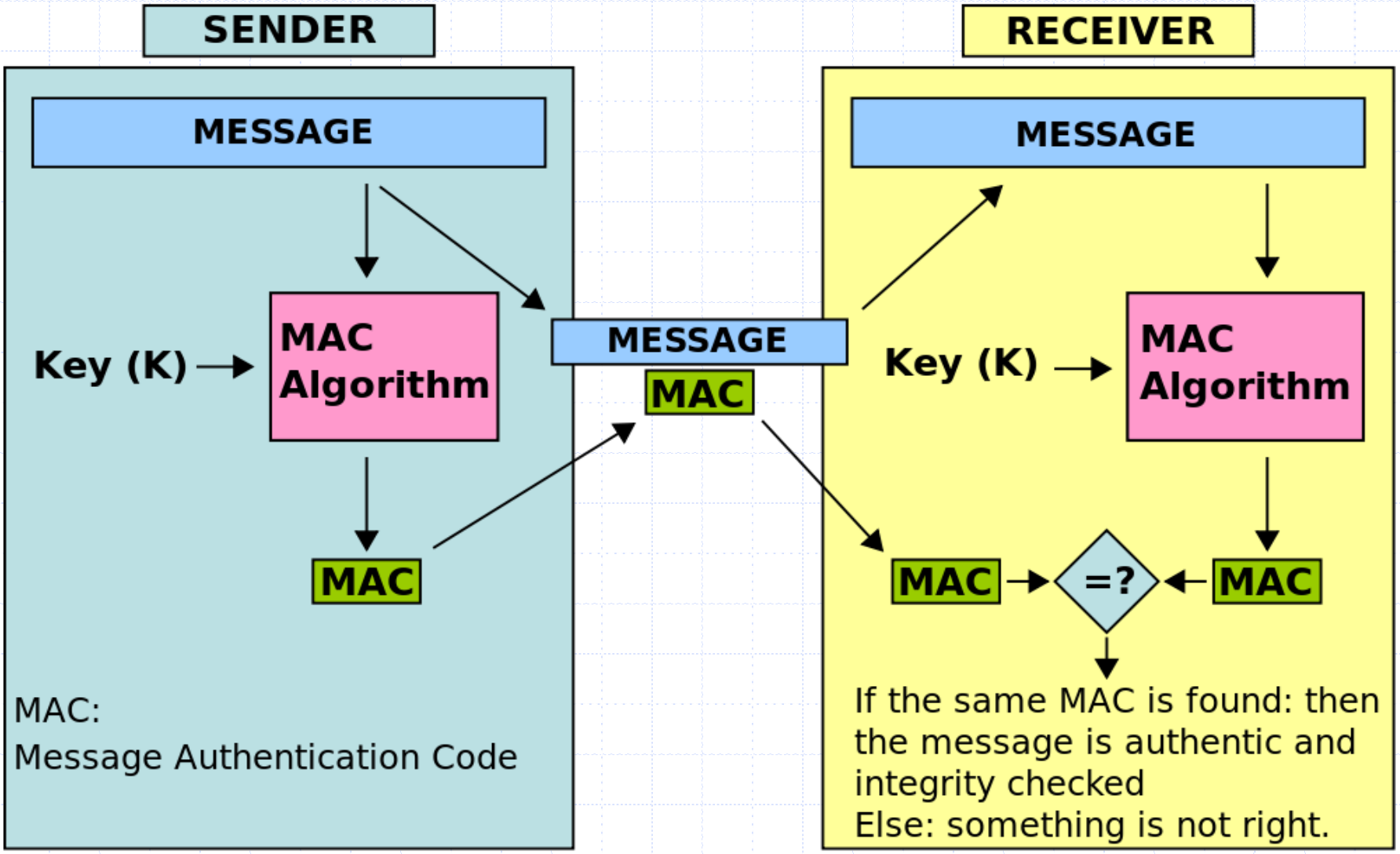


Generate tag:
tag ← S(k, m)

Verify tag: ?
V(k, m, tag) = `yes'

note: non-keyed checksum (CRC) is an insecure MAC !!

Recalling MACs: Length Extension Attacks

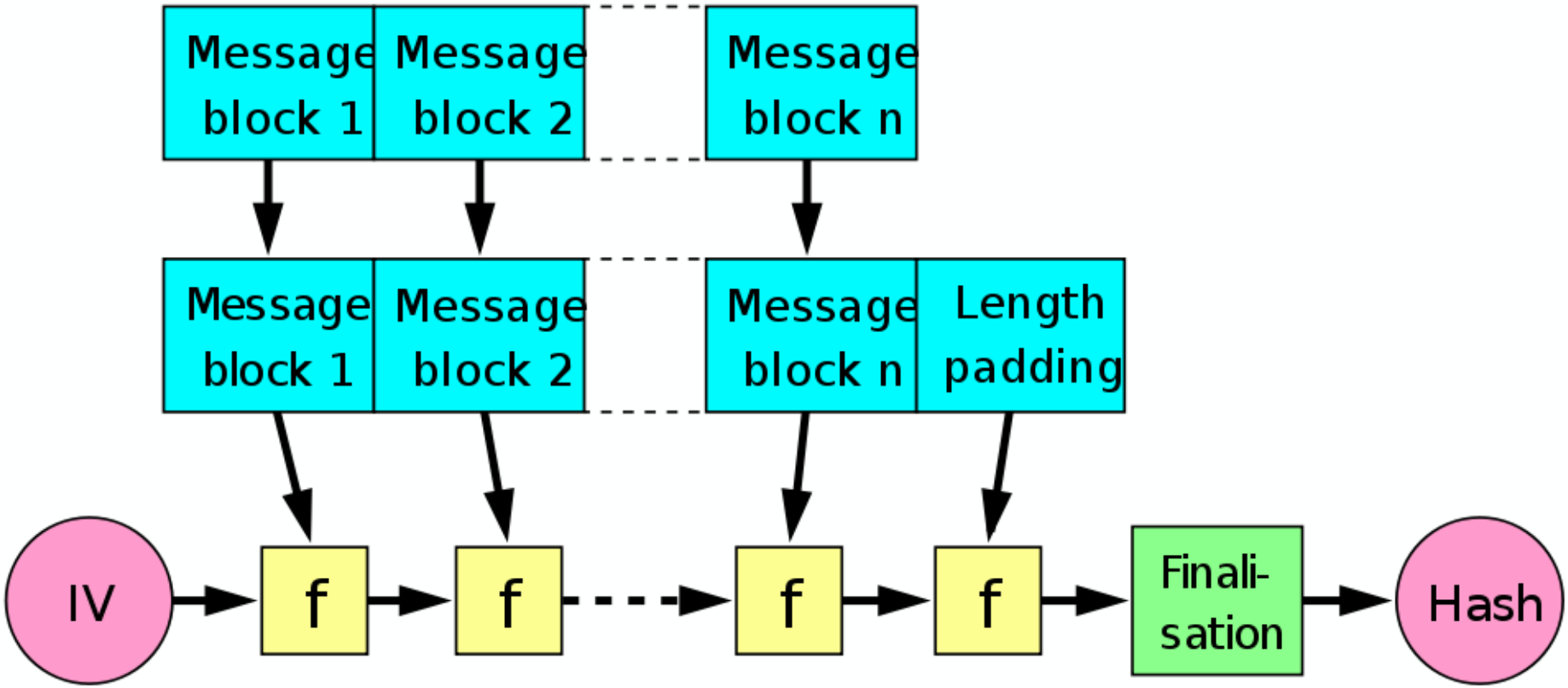


Source: Wikipedia

Length Extension Attacks Explained

- Alice sends "data" and "signature" (the MAC) to Bob. Recall that signature = Hash(secret || data || padding). Padding has a standard format that includes the length of "secret || data"
- Attacker intercepts "data" and "signature"
- Attacker's goal is to append stuff to "data" and appropriately modify signature
- The attacker sends the new "data || attacker extension" and the appropriate "signature" to Bob
- When Bob receives "data || attacker extension" verifies against the new signature, it matches. Attacker doesn't need to know the secret to launch attack. He only needs to know the length of the secret used.

Internals of Hash Functions: Merkle-Damagard Construction



Initialization Vector

Source: Wikipedia

Length Extension Attacks Explained

- Fact:
  - When calculating *H*(*secret* || *data*), the string (*secret* || *data*) is padded with a '1' bit and some number of '0' bits, followed by the length of the string
- Fact:
  - The MD construction operates on fixed-sized blocks, and saves the output for the subsequent iteration. I.e., the signature somehow "captures all of the input data || secret || padding"
- Fact:
  - Attacker knows the data, because Alice sent it along with the signature (MAC)
- We assume attacker knows the length of the secret

Length Extension Attacks Explained

- The Attack:

- Step 1: Compute the padding New-pad for "secret || data || pad (secret || data) || attacker extension"
  - Note that attacker already knows the length of data and length of his own "attacker extension"
  - All he needs is the length of secret to compute New-pad

- Step 2: Attacker initializes the hash function H with "signature" and computes the H(attacker extension || New-pad)

- Step 3: He has a new verifiable signature for New-data = "secret || data || pad(secret + data) || attacker extension || New-pad"

- Step 4: Send the pair <New-data, New-signature> to Bob

MAC Construction: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

H: hash function.

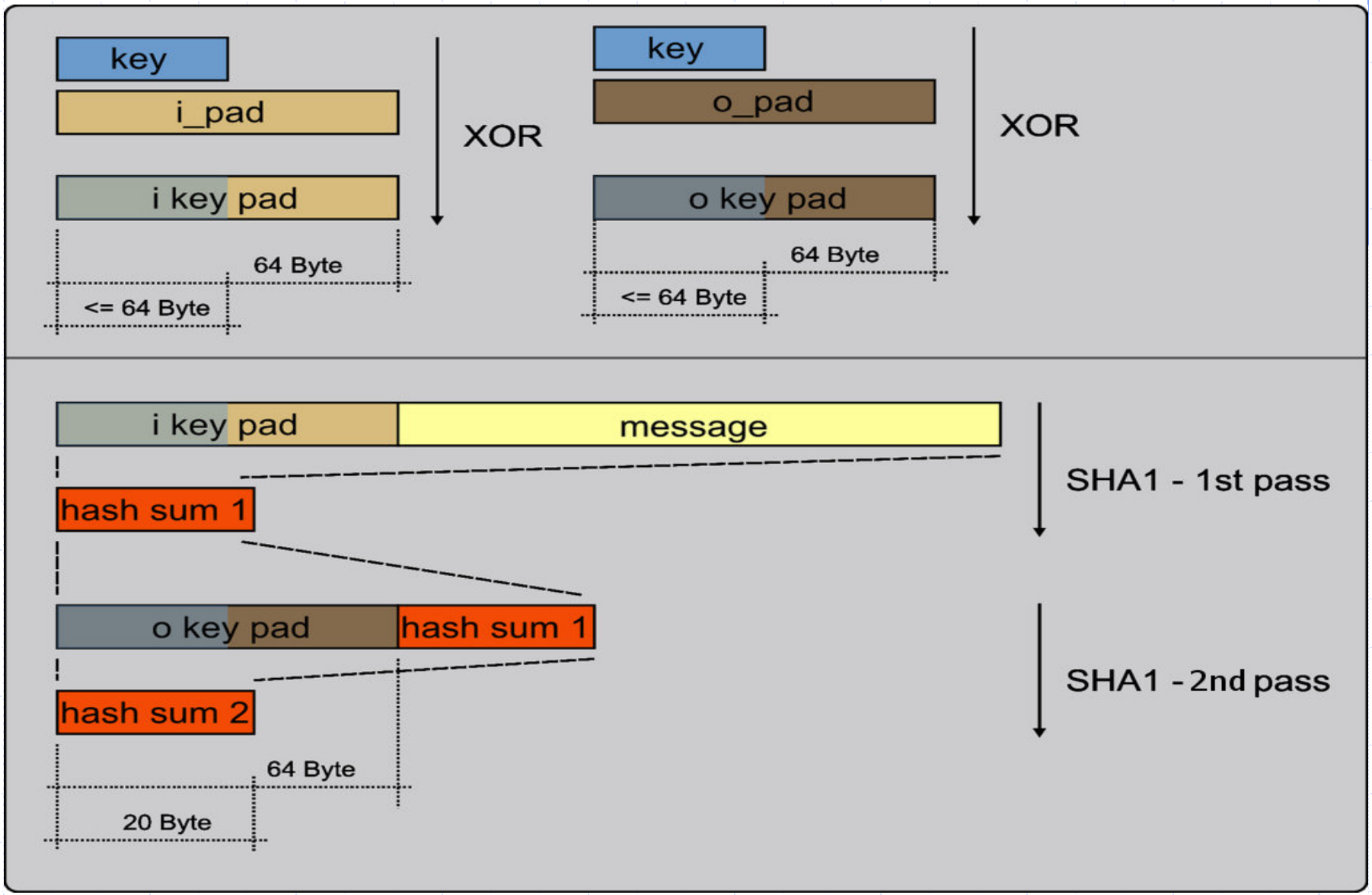
example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

Standardized method: HMAC

$S(k, m) = H(k \oplus opad \,||\, \mathbf{H(k \oplus ipad \,||\, m)})$

Hash MAC (HMAC-SHA256)



Why HMAC is Secure Against Length Extension Attack

- The Attacker knows data, length(secret), hashsum2 (aka signature), and of course his own extension
- The trouble is that he needs to know hashsum1 in order to seed the second call to hash
- Let us try a length extension attack with HMAC:
  - The attacker knows the length of the input to the second hash call, and his own extension
  - Computes new tag by seeding hash call with hashsum2, and hash(attacker extension || New-pad) = signature'
  - This won't verify properly at Bob's end, who is matching signature' ≠ hash(secret || data || attacker-extension)

Use of Cryptographic Salt in Hash-based User Authentication

- Login Program:
  - Computes hash of password, and compares against stored hash. If match, the user is authenticated. Otherwise, authentication attempt is rejected.
- Stored hashes are susceptible to theft
- If passwords are easy, then they are susceptible to dictionary attacks
- Dictionary attacks:
  - Large store of easy passwords
  - Compute hash and compare against stolen stored hashes

Use of Cryptographic Salt in Hash-based User Authentication

- Login Program:
  - Computes hash of password + random seed, and compares against stored hash. If match, the user is authenticated. Otherwise, authentication attempt is rejected.
- Sometimes store even hash(intermediate hashes, password, salt)
- Forces attacker who doesn't know the salt a priori to compute all possible "easy password + salt" combinations
- Modern systems use salts upto 128 bits long
  - Infeasible for attackers to store that large a dictionary

Cryptography Module: Putting it All Together

- Which security problems can cryptography help to solve?
  - Confidentiality through encryption schemes
  - Integrity and Authenticity through MACs and digital signatures
  - User authentication through hash functions
- We studied two forms of encryption schemes
  - Symmetric: Parties must share same key
    - One-time Pad
    - Stream and Block Ciphers
  - Asymmetric: Parties need not share the same key
    - Motivation: Parties don't want to secretly share keys ahead of time
    - RSA public-key encryption

Cryptography Module: Putting it All Together

- Which security problems can cryptography help to solve?
  - Confidentiality through encryption schemes
  - Integrity and Authenticity through MACs and digital signatures
  - User authentication through hash functions
- Digital signature schemes
  - Motivation: Parties want to authenticate messages
  - RSA public-key digital signature schemes
- Hash functions
  - User authentication
  - MACs
  - Integrity