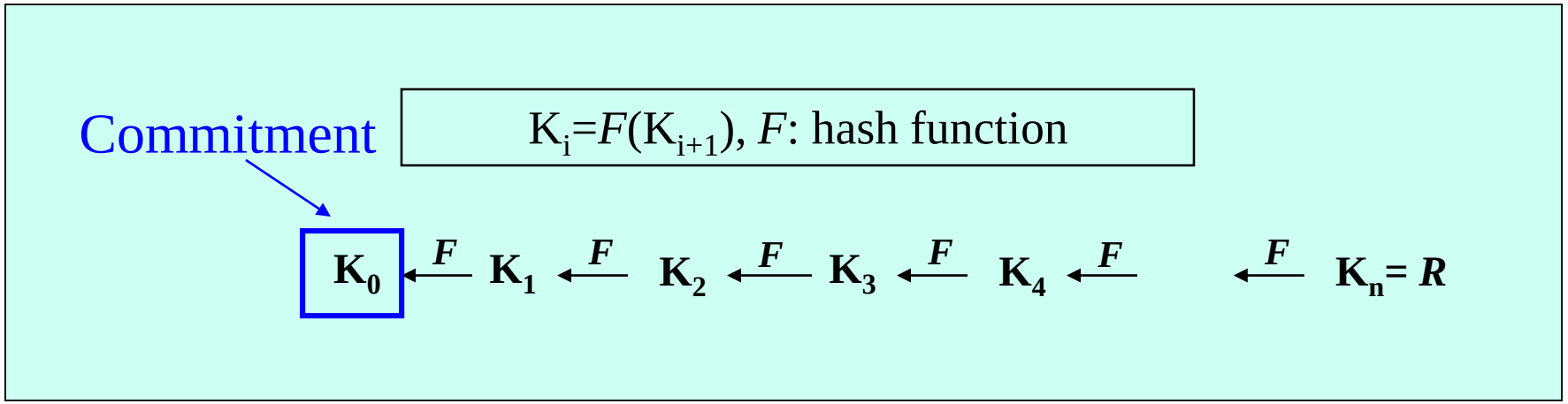
CIS 6930– Emerging Topics in Network Security

Topic 2.2 Hash-based Primitives

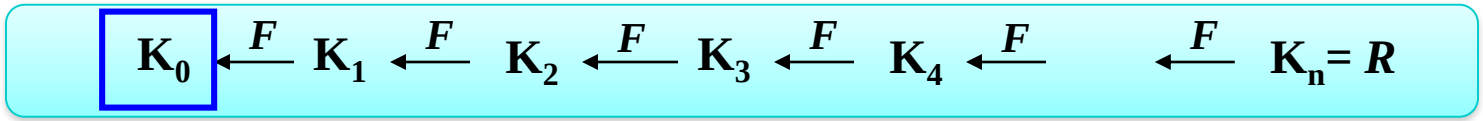
One-way Hash Chain

- Used for many network security applications
  - Example: S/Key
- Good for authentication of the hash values



Properties of One-way Hash Chain

- Given K_i
  - Anybody can compute K_j, where $j<i$
  - It is computationally infeasible to compute K_j, where $j > i$, if K_j is unknown
  - Any K_j disclosed later can be authenticated by verifying if $H^{j-i}(K_i) = K_j$
  - Disclosing of K_{i+1} or a later value authenticates the owner of the hash chain



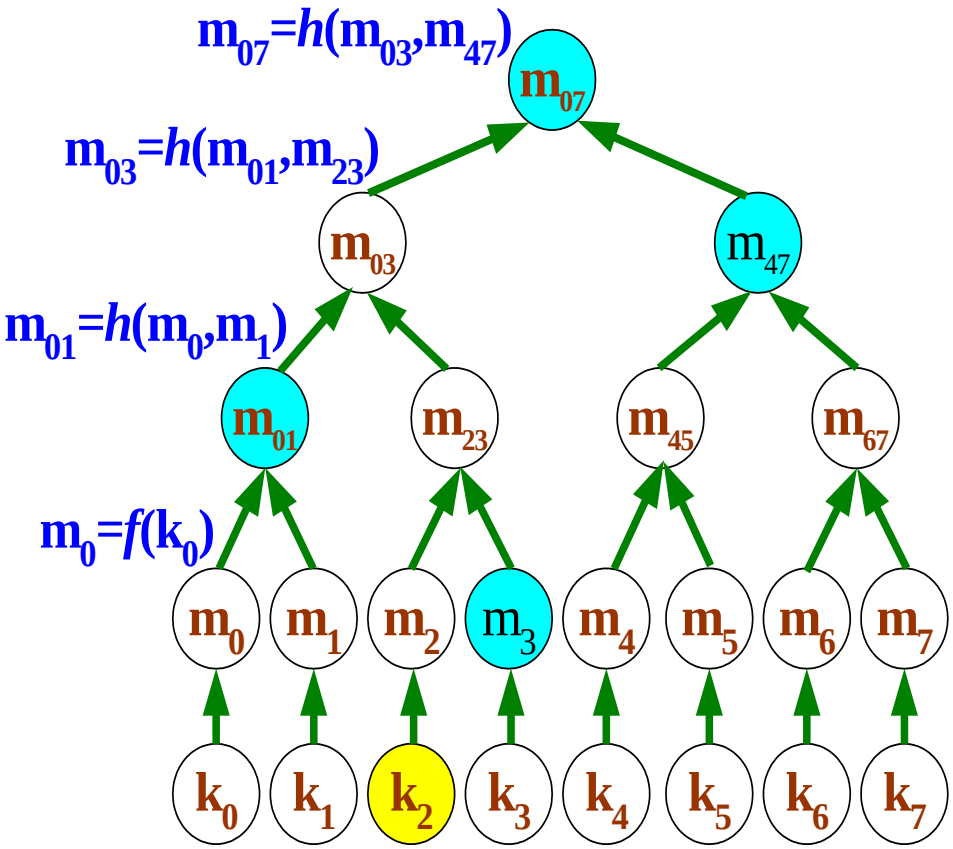
In-class exercise

- Alice sends *n* messages to Bob, and Bob wants to verify that each received message is from Alice.
  - How can the following cryptography tools achieve this goal?
    - Public key cryptography
    - One way hash chain
    - Secret key cryptography
    - Hash function and a shared secret

Merkle Hash Tree

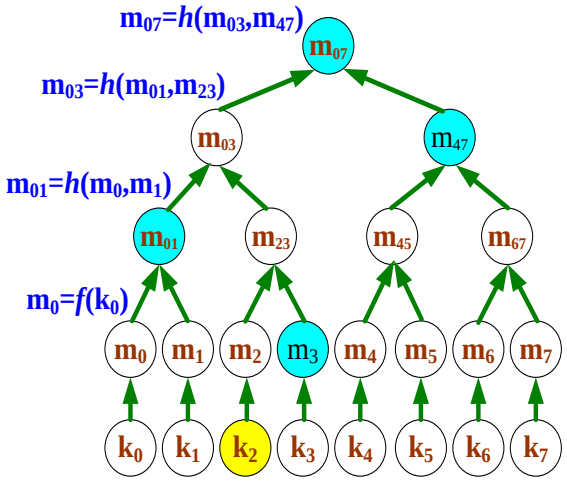
- A binary tree over data values
  - For authentication purpose
- The root is the commitment of the Merkle tree
  - Known to the verifier.
- Example
  - To authenticate k_2, send (k_2, m_3,m_{01},m_{47})
  - Verify

$m_{07}= h(h(m_{01}||h(f(k_2)||m_3)||m_{47})$



Merkle Hash Tree (Cont'd)

- Hashing at the leaf level is necessary to prevent unnecessary disclosure of data values
- Authentication of the root is necessary to use the tree
  - Typically done through a digital signature or pre-distribution



In-class exercise

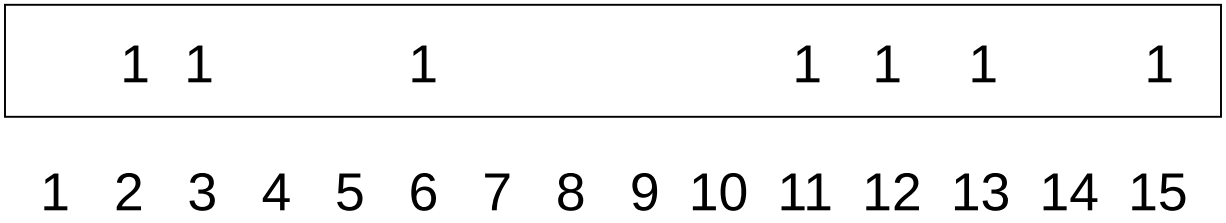
- Alice sends *n* messages to Bob, and Bob wants to verify that each received message is from Alice
  - How can Merkle hash tree achieve this goal?
  - How can Merkle hash tree protect data integrity?

Bloom Filters

- It is used to verify that some data is *not* in the database (mismatch)
  - List of bad credit card numbers
  - Useful when the data consumes a very small portion of search space
- A bloom filter is a bit string
- *k* hash functions that map the data into *n* bits in the bloom filter

A Simple Example

- Use a bloom filter of 16 bits
  - $H_1(key) = key \bmod 16$
  - $H_2(key) = key \bmod 14 + 2$
- Insert numbers 27, 18, 29 and 28



- Check for 22:
  - $H_1(22) = 6$, $H_2(22) = 10$ (not in filter)
- Check for 51
  - $H_1(51) = 3$, $H_2(51) = 11$ (false positive)

Probability of False Positive

- Consider an *m*-bit Bloom filter with *k* hash functions
  - After inserting *n* bits, the probability of false positive

$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

- What is the probability of false negative?

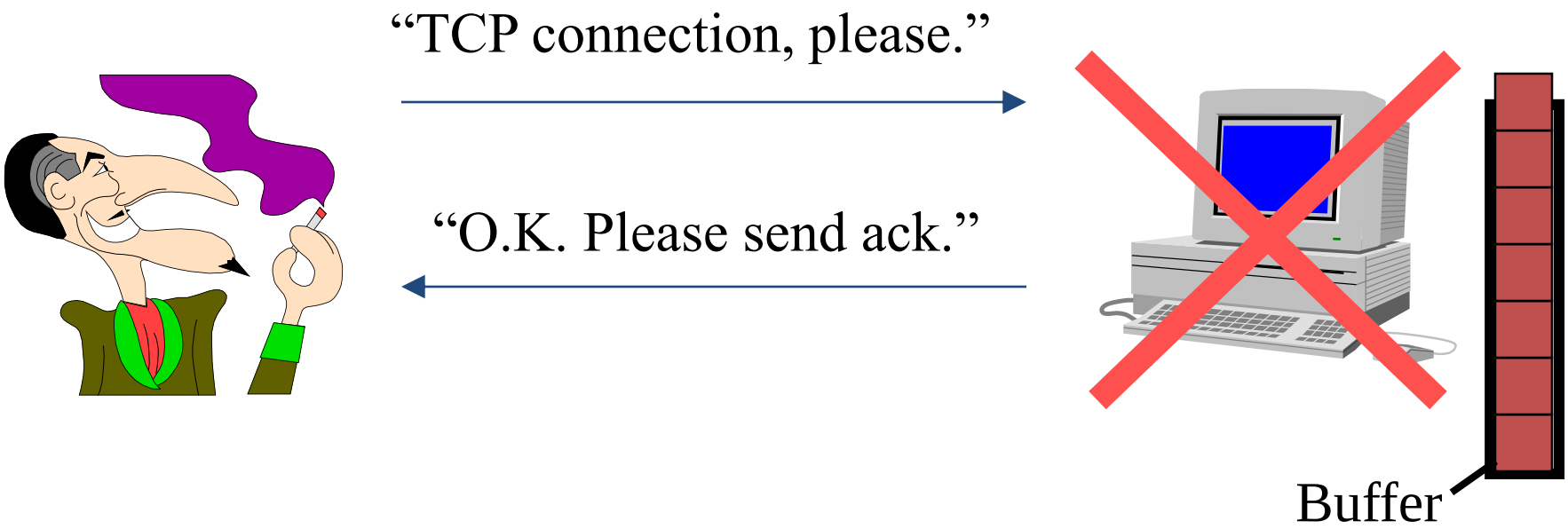
Client Puzzles

- The problem being addressed
  - Denial of Service (DoS) attacks
- Three basic constructions
  - Use pre-image of crypto hash functions
  - Use special image of crypto hash functions

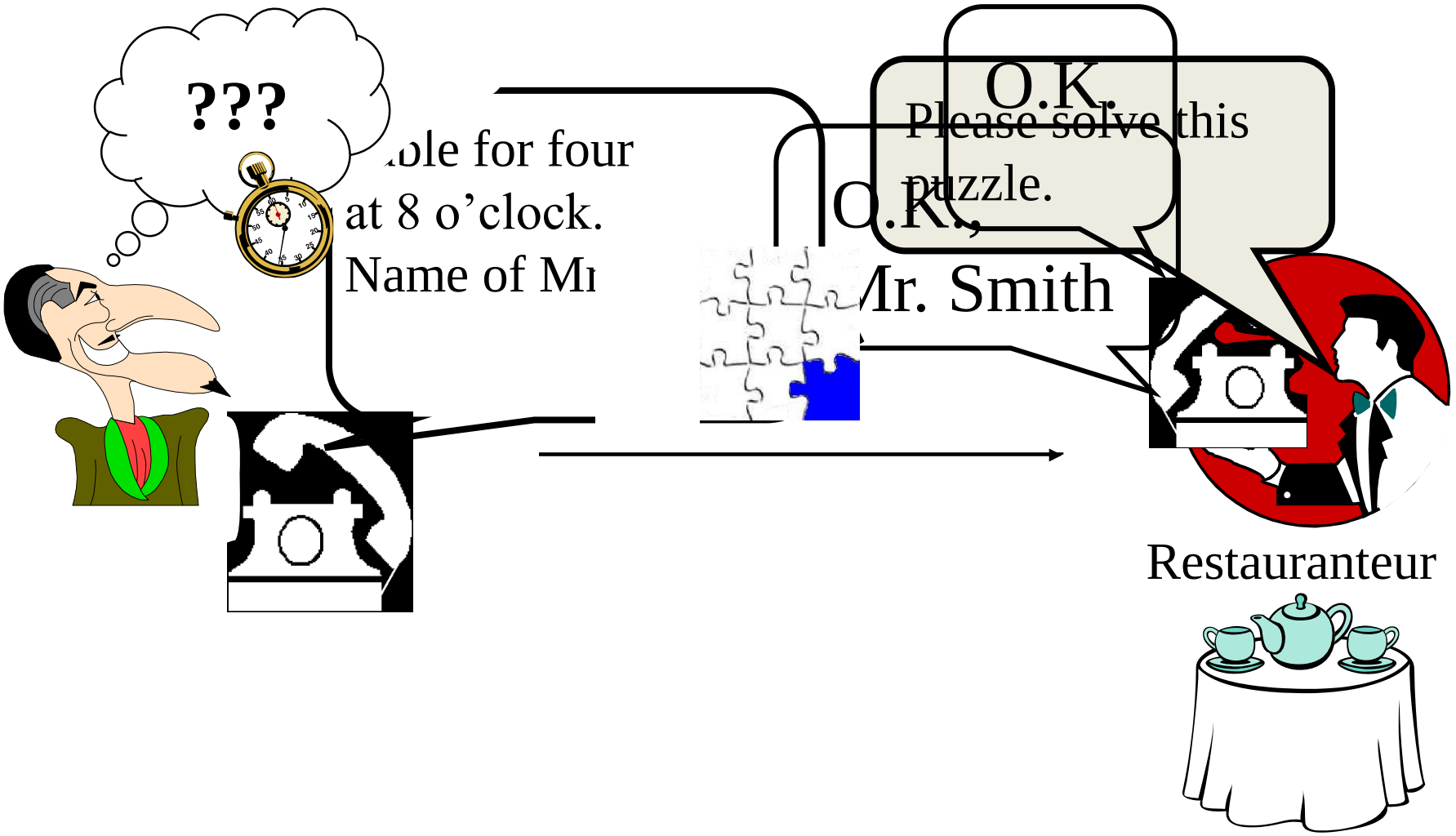
Client Puzzle Based on Pre-image of Crypto Hash Functions

Slides modified from those by Ari Juels and John Brainard

An Example Scenario: TCP SYN Flooding



Client Puzzle: Intuition

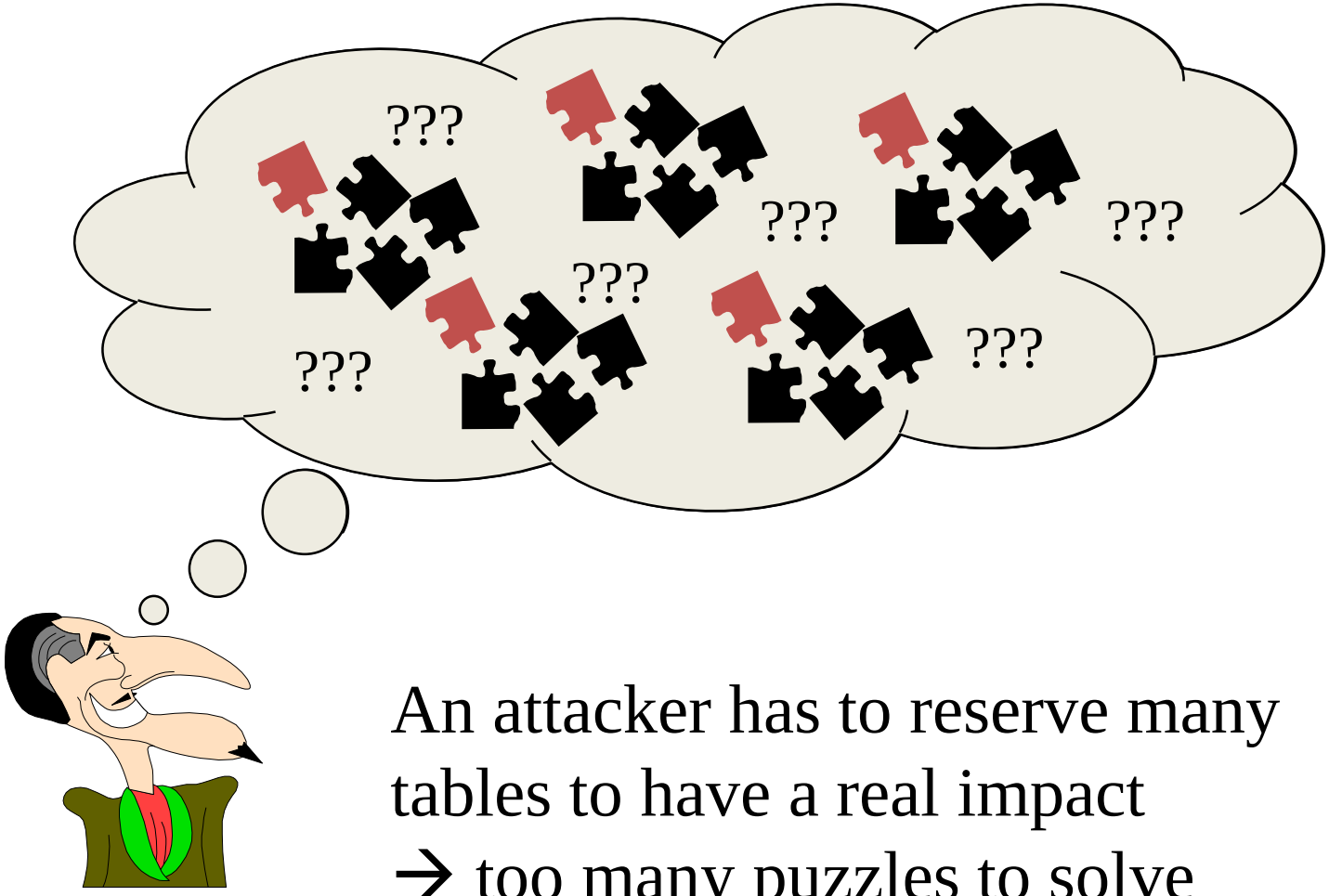


Client Puzzle: Intuition

- A puzzle takes an hour to solve
- There are 40 tables in restaurant
- Reserve at most one day in advance

A legitimate patron can easily reserve a table

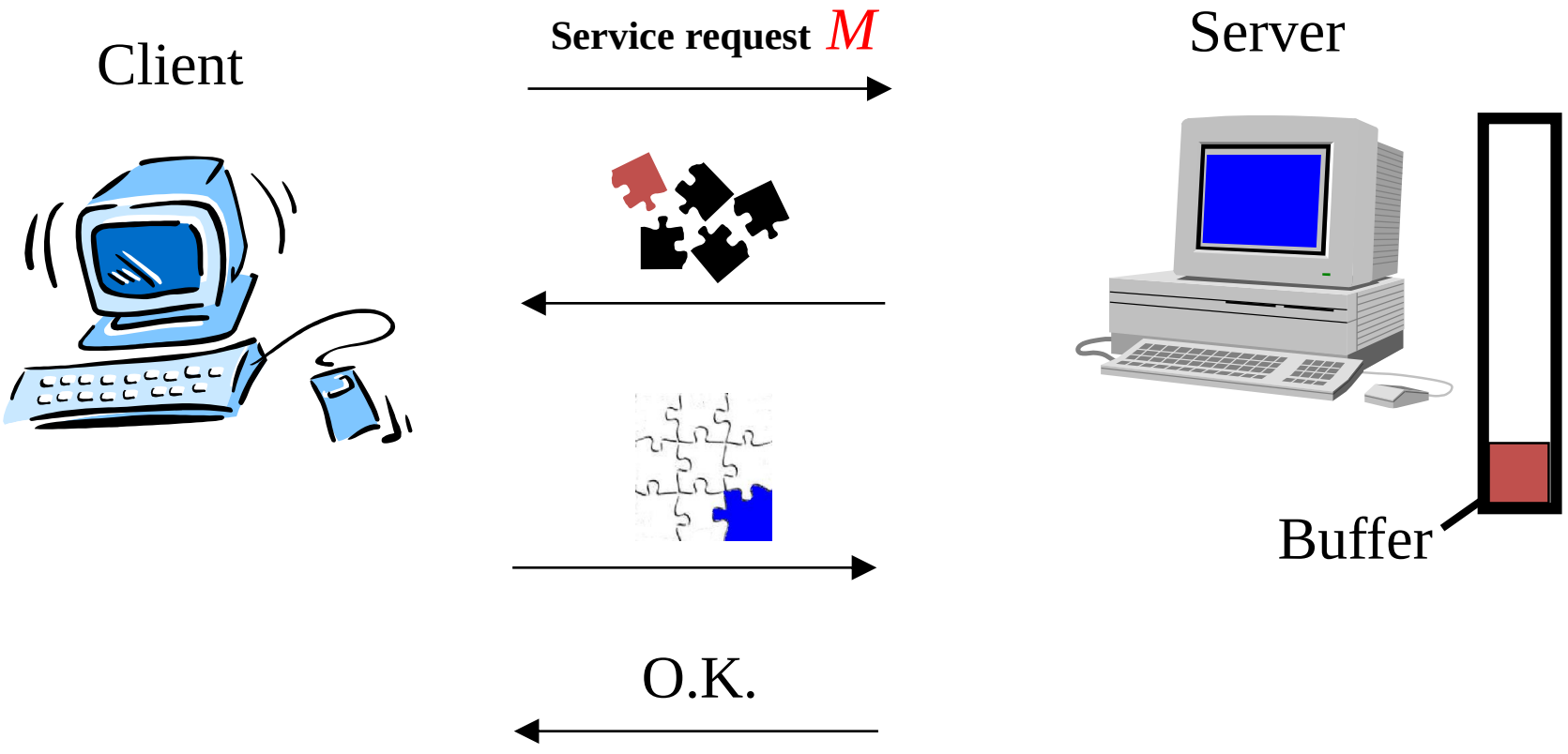
Client Puzzle: Intuition



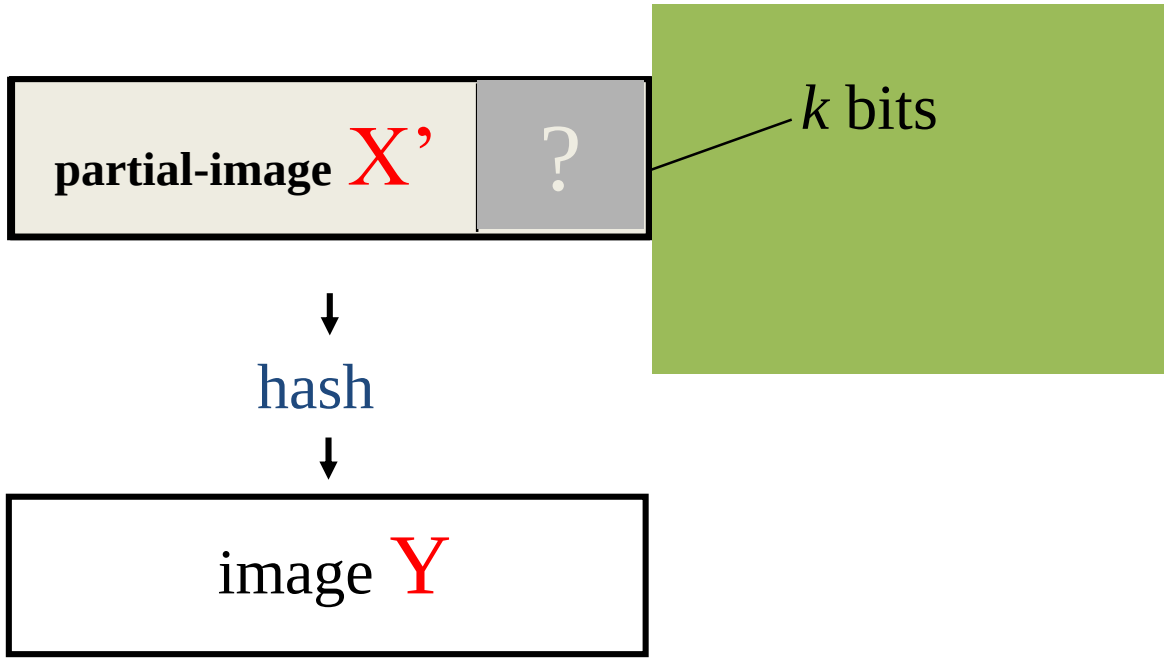
An attacker has to reserve many tables to have a real impact

→ too many puzzles to solve

The Client Puzzle Protocol



Puzzle Basis: Partial Hash Image



Pair (X', Y) is *k*-bit-hard puzzle

Puzzle Basis (Cont'd)

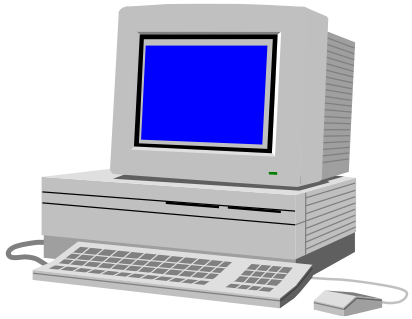
- Only way to solve puzzle (X',Y) is brute force method. (hash function is not invertible)

Puzzle Construction

Client

Server

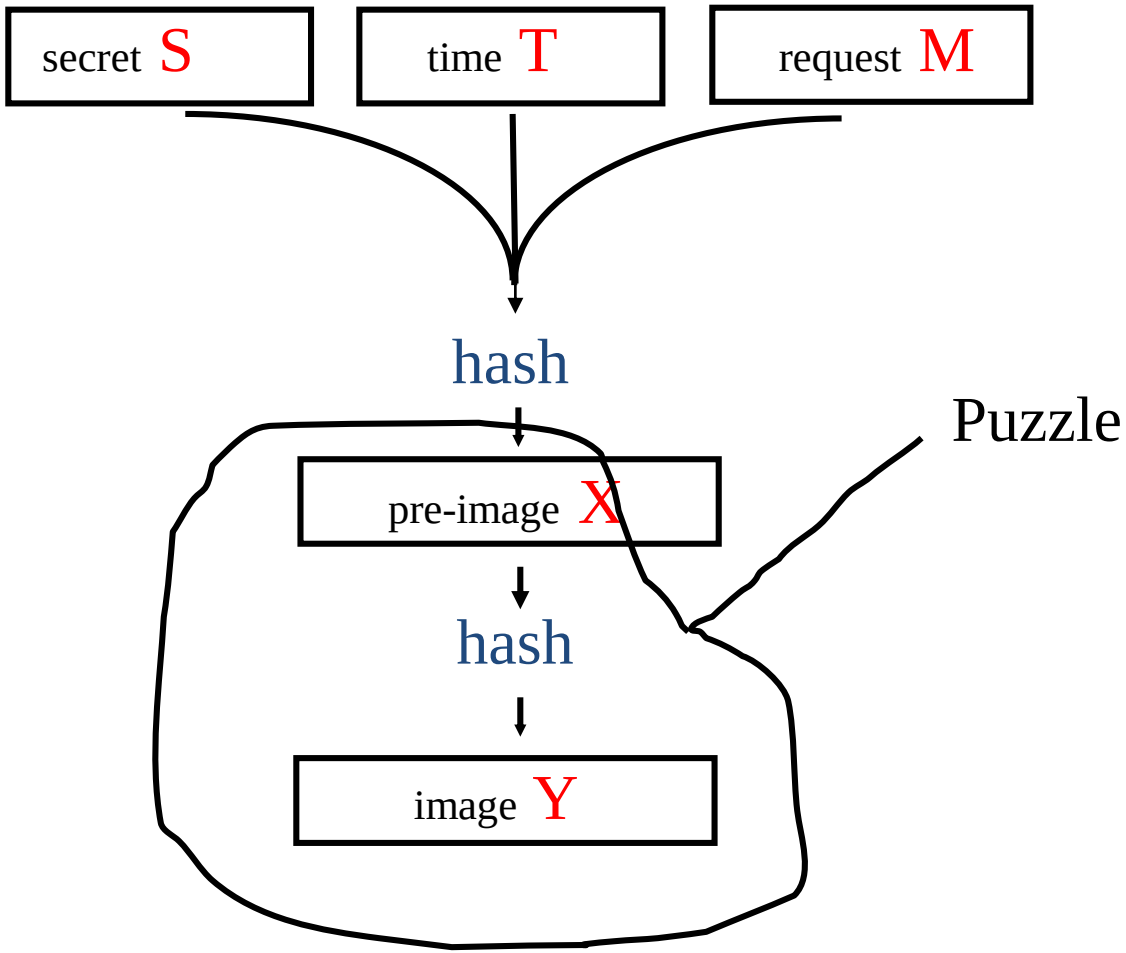
Service request M



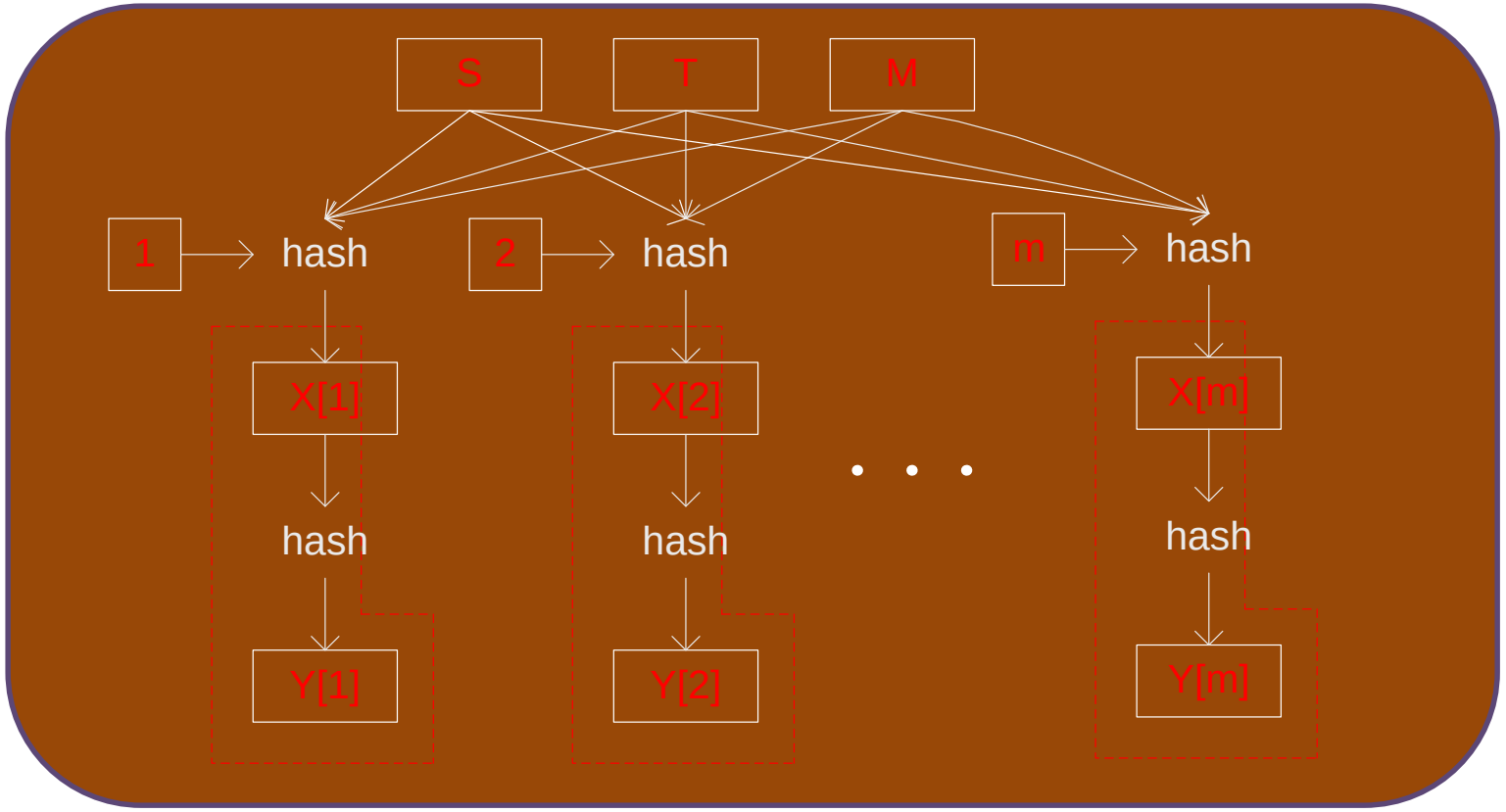
Secret S

Puzzle Construction

Server computes:



Sub-puzzle



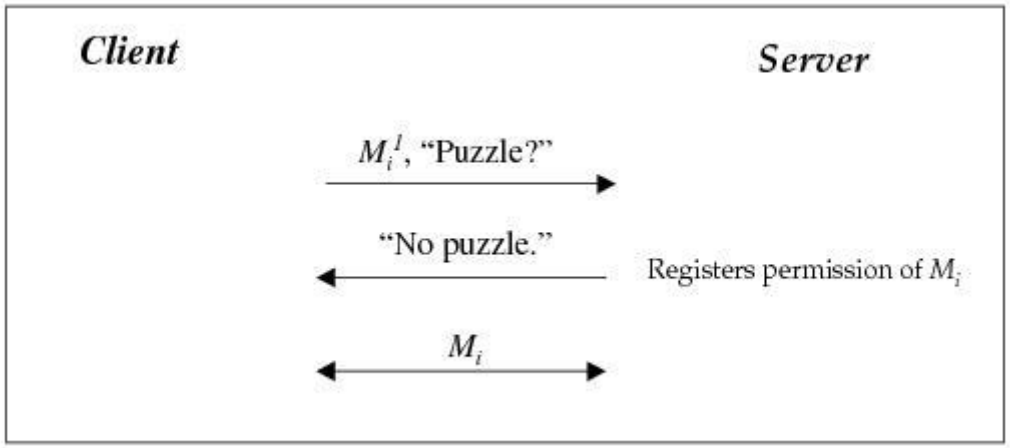
- Construct a puzzle consisting of *m* k-bit-hard sub-puzzles.
- Increase the difficulty of guessing attacks.

Puzzle Properties

- Puzzles are stateless
- Puzzles are easy to verify
- Hardness of puzzles can be carefully controlled
- Puzzles use standard cryptographic primitives

A Possible Way to use Client Puzzle

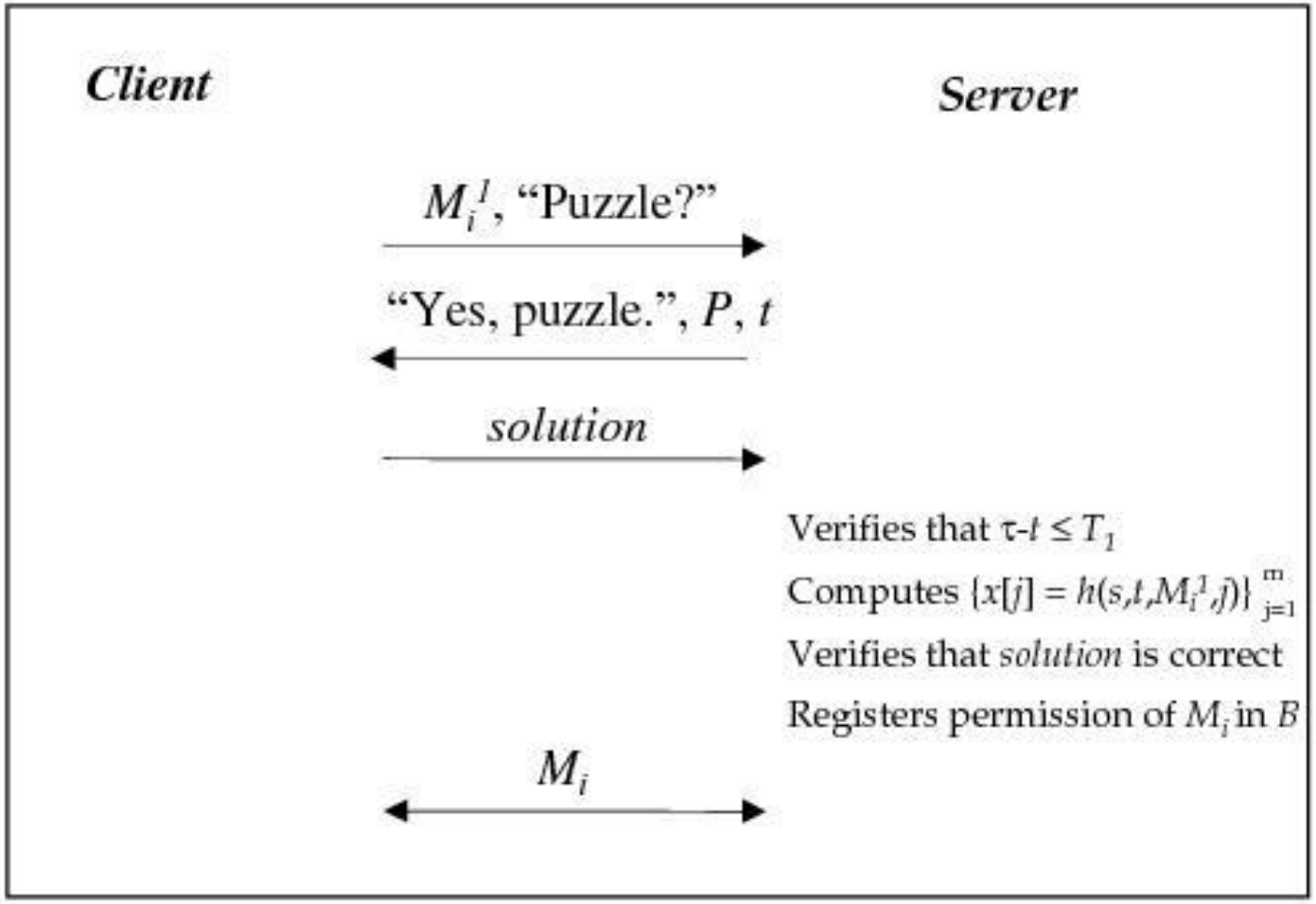
Client puzzle protocol (normal situation)



M_i^1 : first message of i-*th* execution of protocol M

A Possible Way to use Client Puzzle

Client puzzle protocol (under attack)



Client Puzzle Based on Special Image of Crypto Hash Functions

Puzzle Construction

- C $\rightarrow$ S: Hello
- S $\rightarrow$ C: N_S
- C $\rightarrow$ S: C, N_S, N_C, X
- S: verify $h(C, N_S, N_C, X)$ has k leading zero's

$$h(C, N_S, N_C, X) = \overbrace{000\ldots000}^{\text{the } k \text{ first bits of the hash}} \underbrace{Y}_{\text{the rest of the hash bits}}$$

h	=	a cryptographic hash function (e.g. MD5 or SHA)
C	=	the client indentity
N_S	=	the server's nonce
N_C	=	the client's nonce
X	=	the solution of the puzzle
k	=	the puzzle difficulty level
$000\ldots000$	=	the k first bits of the hash value; must be zero
Y	=	the rest of the hash value; may be anything

Expected Cost of Finding a Puzzle Solution

- Let *n* be the length of the puzzle solution, the probability of finding a solution
  - After *x* trials:

$$P_{\geq x,n} = 1 - (1 - 2^{-n})^{x}$$

  - At the *x-th* trial:

$$P_{x,n} = 2^{-n}(1 - 2^{-n})^{x-1}$$

- Expected number of trials to find a solution is 2^n