

# Cryptography Basis and Applications

**MAO Jian**

[maojian@buaa.edu.cn](mailto:maojian@buaa.edu.cn)

Beihang University

School of Cyber Science and Technology

Credits: Some slides from David Brumley's Introduction to Cryptography

Textbook: Information Security Principle and Practice (Second edition), Mark Stamp

# Lecture Plan

- Basis of Cryptography
  - Concepts
  - Symmetric Key Crypto System
  - Public Key Crypto System
- Applications of Crypto Systems
  - Hash Function
  - Password-based Authentication
  - Secure Socket Layer (SSL)
- Labs

# The Cast of Characters

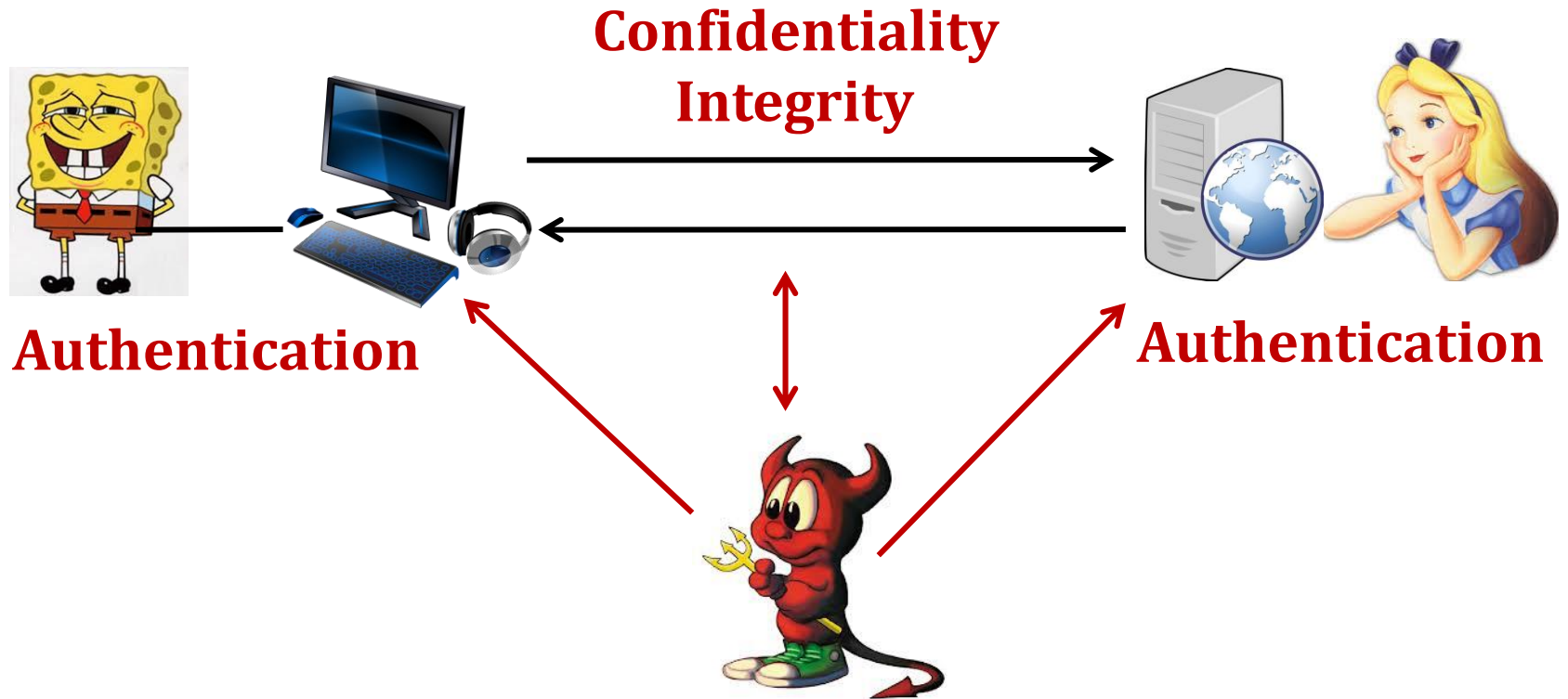
- Alice and Bob are the good guys



- Trudy is the bad “guy”
  - Trudy is our generic “intruder”



# How to secure a web session?



# Cryptography is Everywhere

## Secure communication:

- web traffic: HTTPS
- wireless traffic: 802.11i WPA2 (and WEP), GSM, Bluetooth

## Encrypting files on disk: EFS, TrueCrypt

## Content protection:

- CSS (DVD), AACS (Blue-Ray)

## User authentication

- Kerberos, HTTP Digest

***... and much much more***

# Babylon's Words

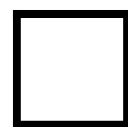
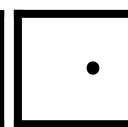
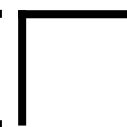



ab	cd	ef
gh	ij	kl
mn	op	qr

uv  
st wx  
yz

a →  b → 

***b a b y l o n***



# Notions & Key Terms

Crypto, Cryptography

Crypto System

Crypto Assumption: Kerchoffs's Principle

# Crypto

In the early age,

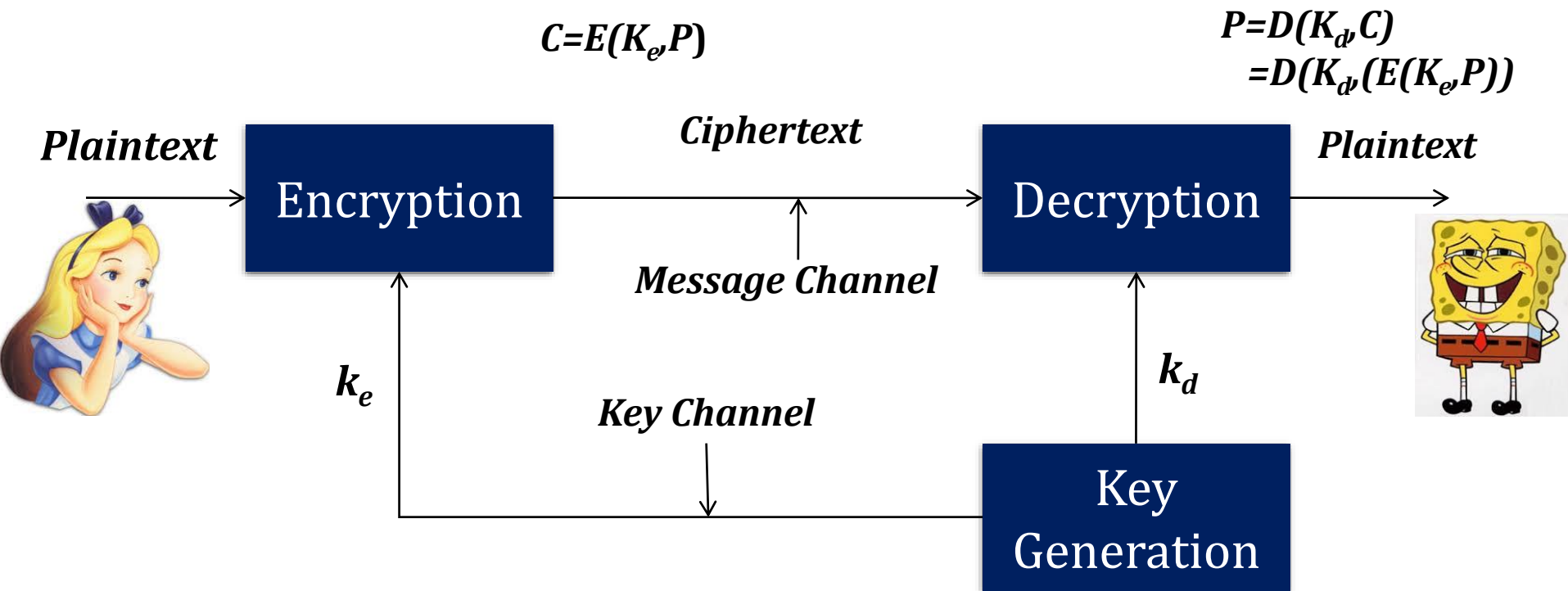
**“Information Security” = “Crypto”**

- **Cryptology** — The art and science of making and breaking “secret codes”
- **Cryptography** — making “secret codes”
- **Cryptanalysis** — breaking “secret codes”
- **Crypto** — all of the above (and more)

**Kerckhoffs’s Principle:** the system is completely known to the attacker; only the key is secret



# Cryptographic systems (P,C,E,D,K)



- Secret-Key Cryptosystem:  $k_e = k_d$
- Public-Key Cryptosystem:  $k_e \neq k_d$

Key channel: e.g. Courier  
Key channel: e.g. Directory

# Taxonomy of Cryptography

- **Symmetric Key**

- Same key for encryption and decryption
- Two types: Stream ciphers, Block ciphers

- **Public Key** (or asymmetric crypto)

- Two keys, one for encryption (public), and one for decryption (private)
- Digital signatures — nothing comparable in symmetric key crypto

- **Hash algorithms**

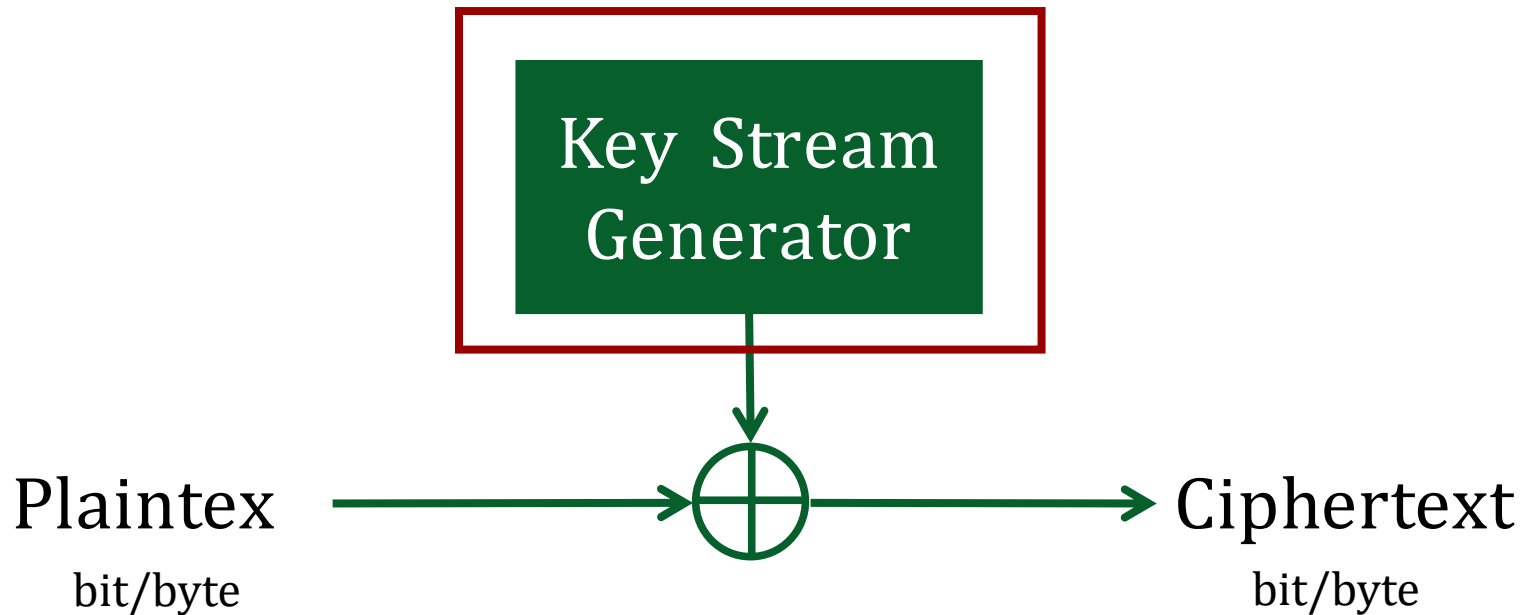
- Can be viewed as “one way” crypto

# Symmetric Key Crypto System

Stream Cipher

Block Cipher

# What is a Stream Cipher?

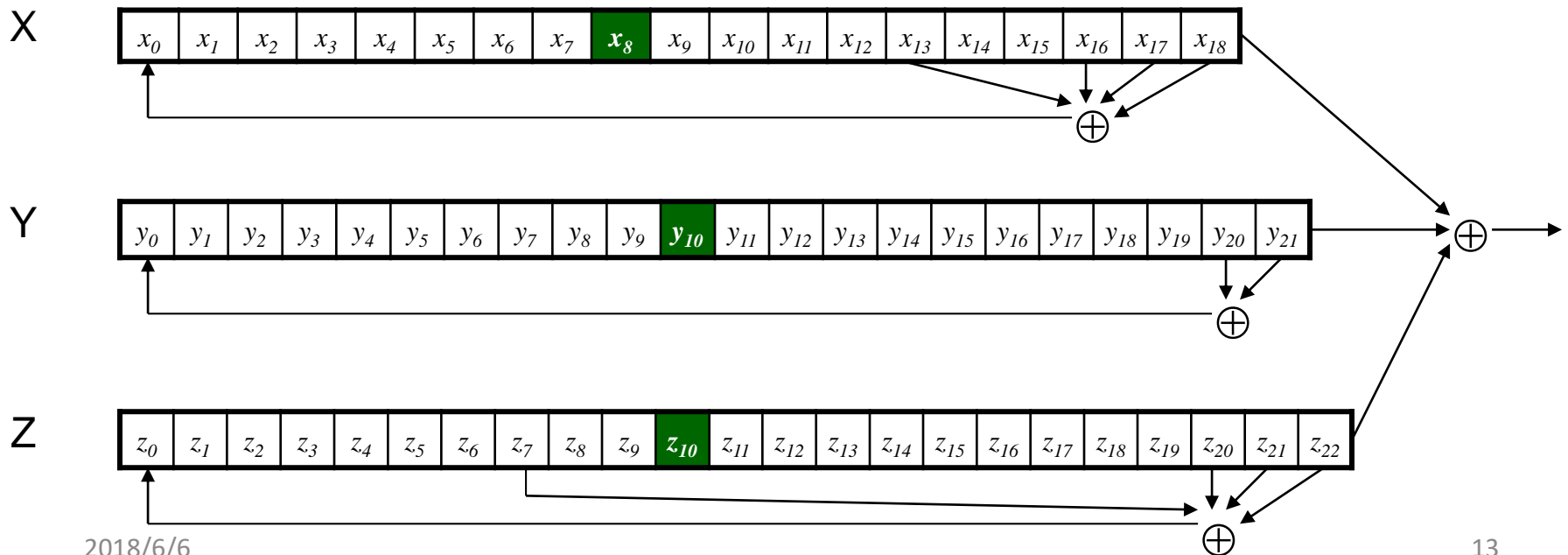


## Canonical examples:

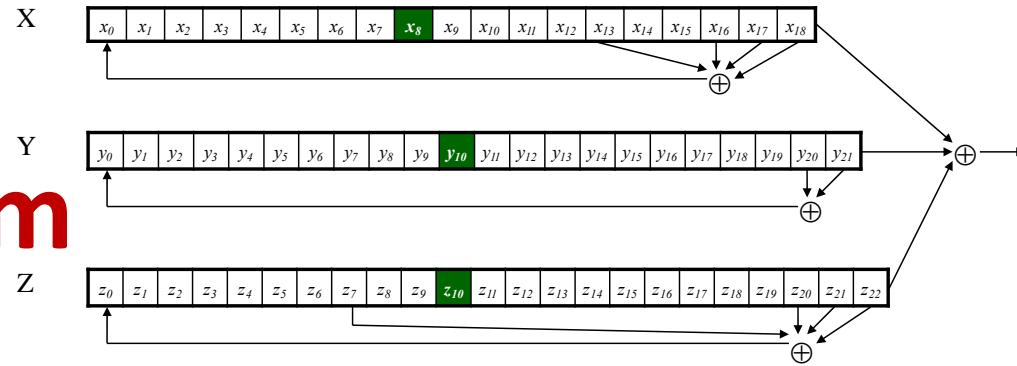
1. A5/1—GSM
2. RC4 –SSL/TLS, WEP, WPA...

# A5/1: Shift Registers

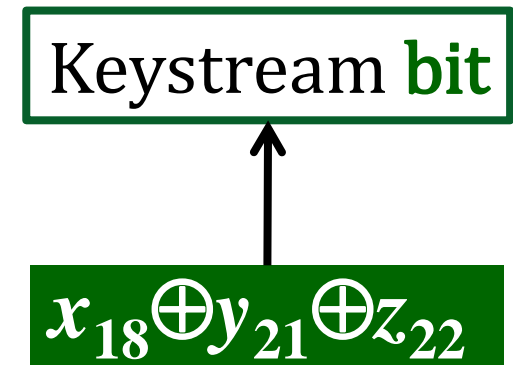
- A5/1 uses 3 *shift registers*
  - X: 19 bits ( $x_0, x_1, x_2, \dots, x_{18}$ )
  - Y: 22 bits ( $y_0, y_1, y_2, \dots, y_{21}$ )
  - Z: 23 bits ( $z_0, z_1, z_2, \dots, z_{22}$ )



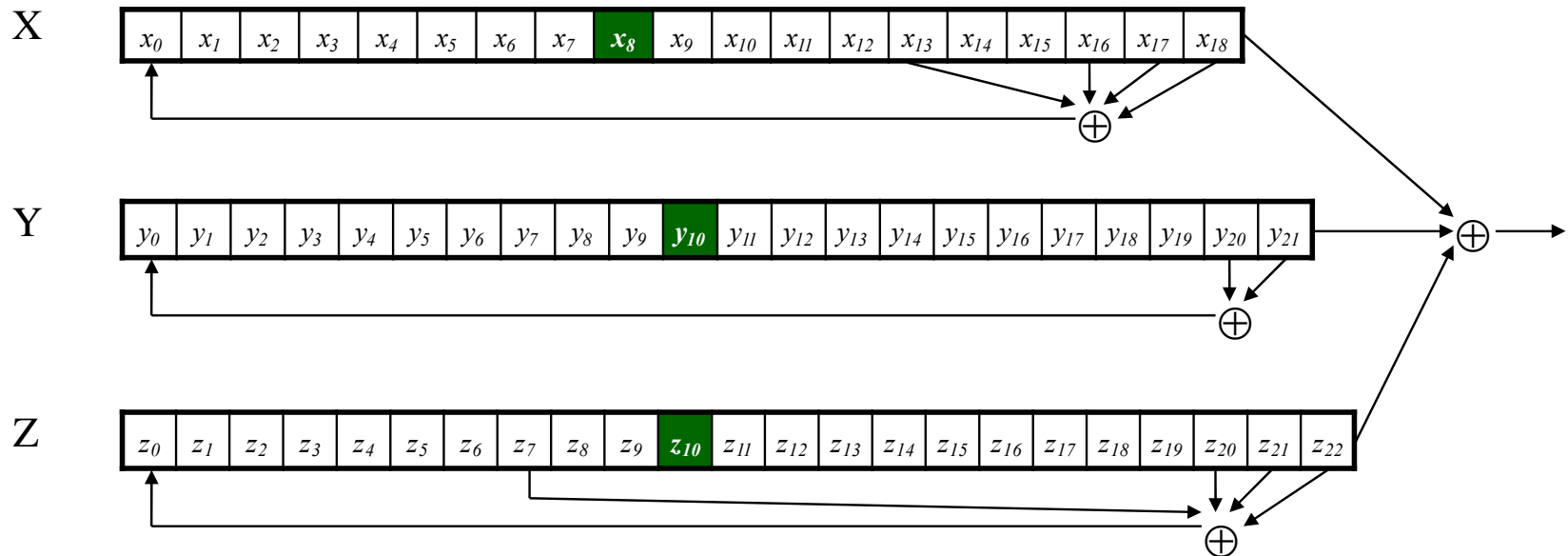
# A5/1: Keystream



- At each step:  $m = \text{maj}(x_8, y_{10}, z_{10})$ 
  - Examples:  $\text{maj}(0,1,0) = 0$   
 $\text{maj}(1,1,0) = 1$
- If  $x_8 = m$  then *X steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$  for  $i = 18, 17, \dots, 1$  and  $x_0 = t$
- If  $y_{10} = m$  then *Y steps*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$  for  $i = 21, 20, \dots, 1$  and  $y_0 = t$
- If  $z_{10} = m$  then *Z steps*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$  for  $i = 22, 21, \dots, 1$  and  $z_0 = t$

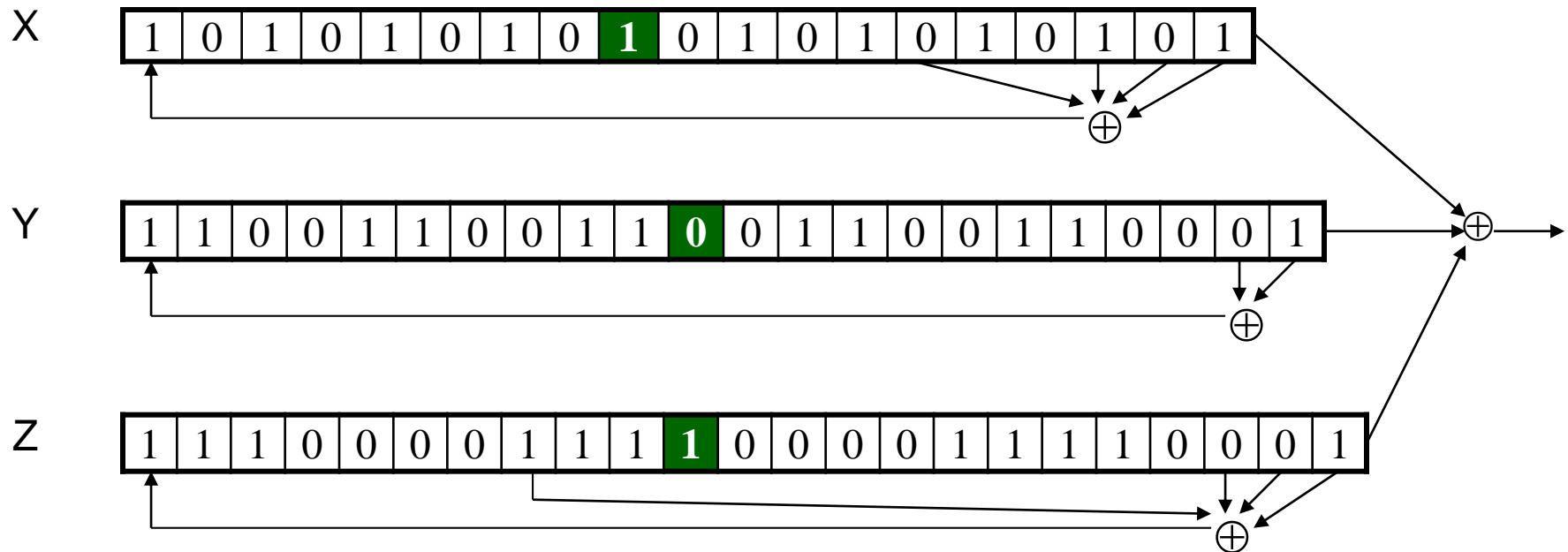


# A5/1



- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on  $\text{maj}(x_8, y_{10}, z_{10})$
- Keystream bit is XOR of rightmost bits of registers

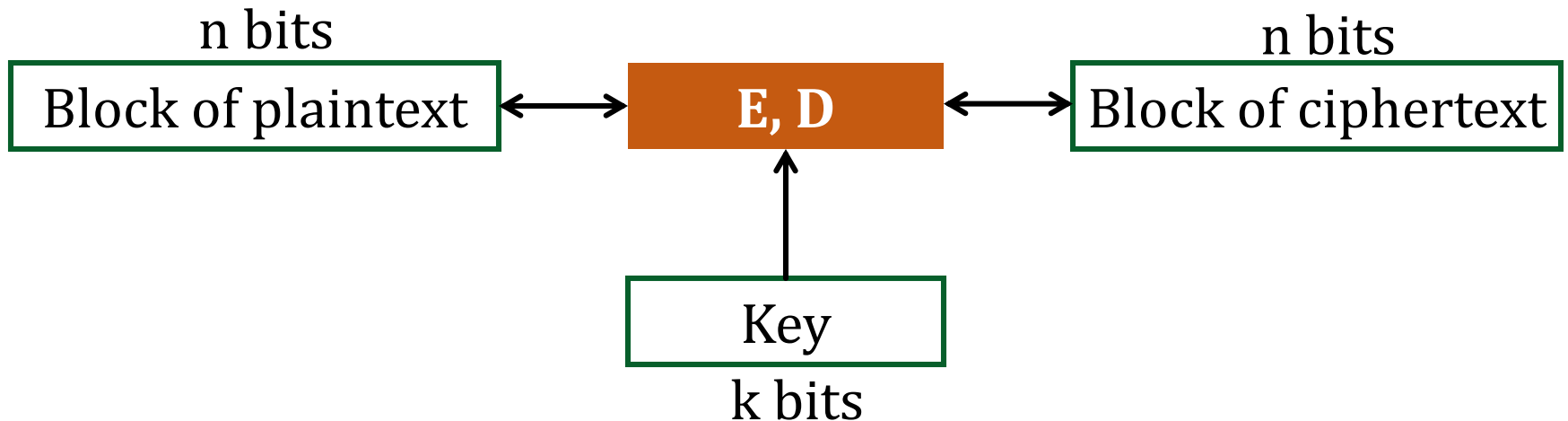
# A5/1



- In this example,  $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(\mathbf{1}, \mathbf{0}, \mathbf{1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be  $\mathbf{0 \oplus 1 \oplus 0 = 1}$



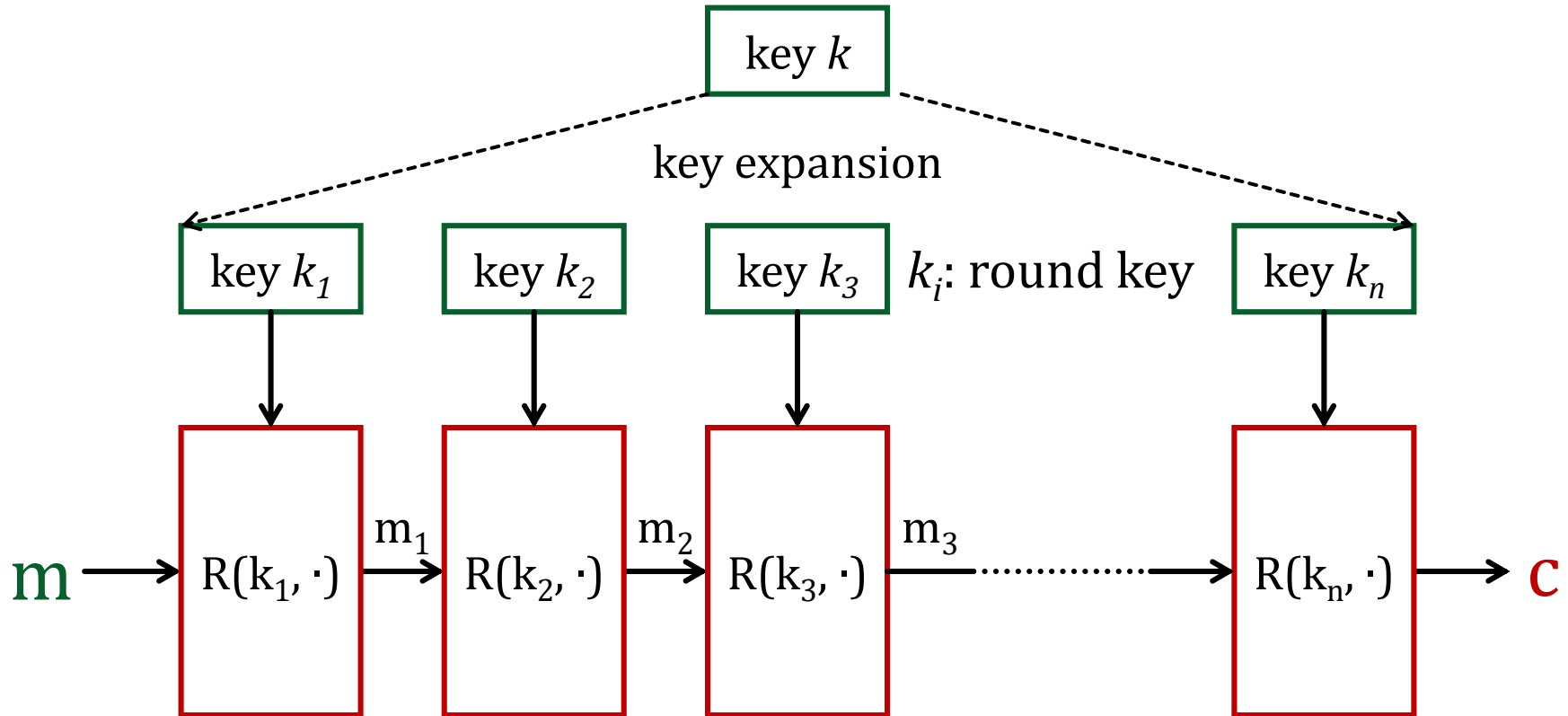
# What is a block cipher?



## Canonical examples:

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

# Block ciphers built by iteration



$R(k, m)$  is called a round function

Ex: DES ( $n=16$ ), AES128 ( $n=10$ )

# Data Encryption Standard (DES)

History

Overall Structure & Algorithm

Security Analysis

Triple DES

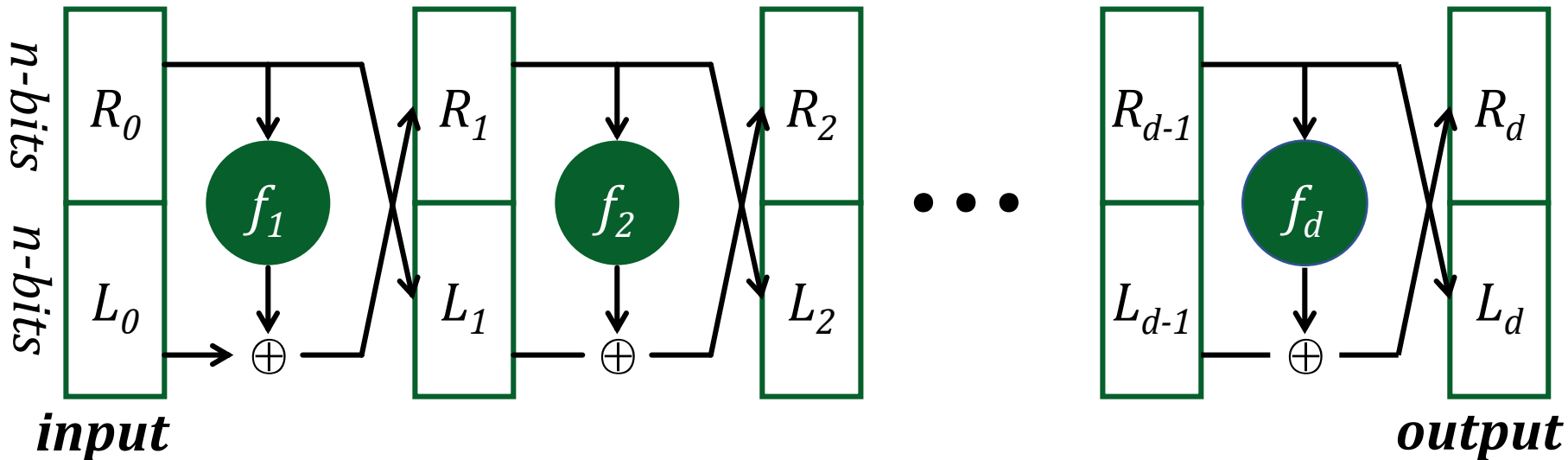
# History of DES

- **1970s:** Horst Feistel designs Lucifer at IBM
  - key = 128 bits, block = 128 bits
- **1973:** NBS asks for block cipher proposals.
  - IBM submits variant of Lucifer.
- **1976:** NBS adopts DES as federal standard
  - key = 56 bits, block = 64 bits
- **1997:** DES is broken by exhaustive search
- **2000:** NIST adopts Rijndael as AES to replace DES.
  - AES currently widely deployed in banking, commerce and Web

# DES: core idea – Feistel network

Given one-way functions  $f_1, \dots, f_d : \{0, 1\}^n \rightarrow \{0, 1\}^n$

Goal: build invertible function  $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$



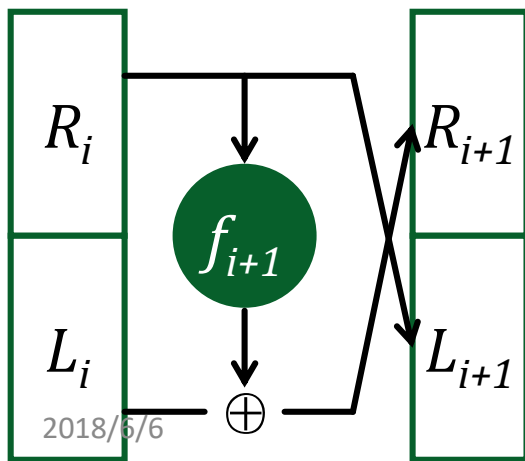
In symbols: 
$$\begin{cases} R_i = f_i(R_{i-1}) \oplus L_{i-1} \\ L_i = R_{i-1} \end{cases}$$

# Feistel network - inverse

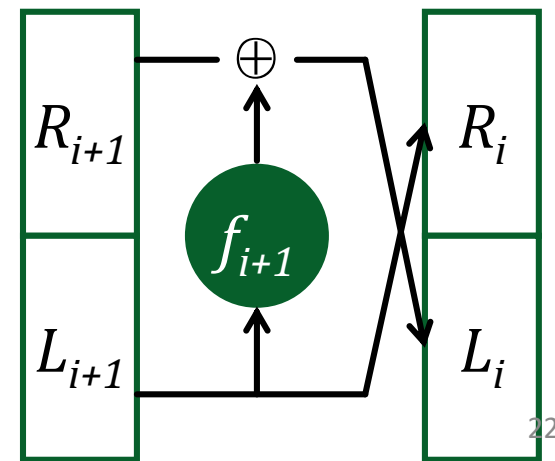
Claim:  $f_1, \dots, f_d : \{0, 1\}^n \rightarrow \{0, 1\}^n$

Feistel function  $F$  is invertible  $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$

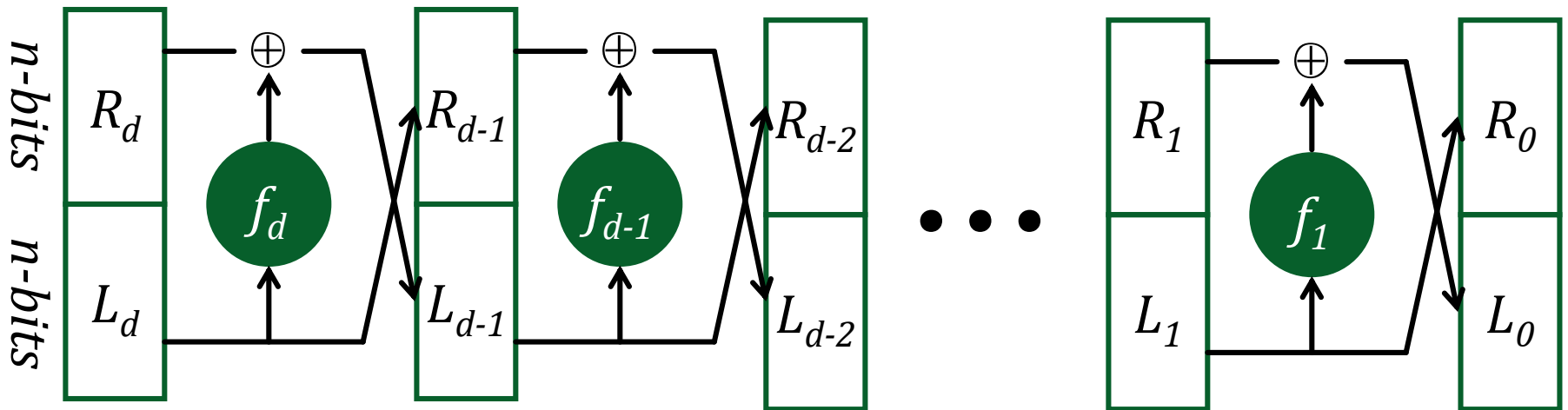
Proof: construct inverse 
$$\begin{cases} R_i = L_{i+1} \\ L_i = R_{i+1} \oplus f_{i+1}(L_{i+1}) \end{cases}$$



inverse



# Decryption circuit

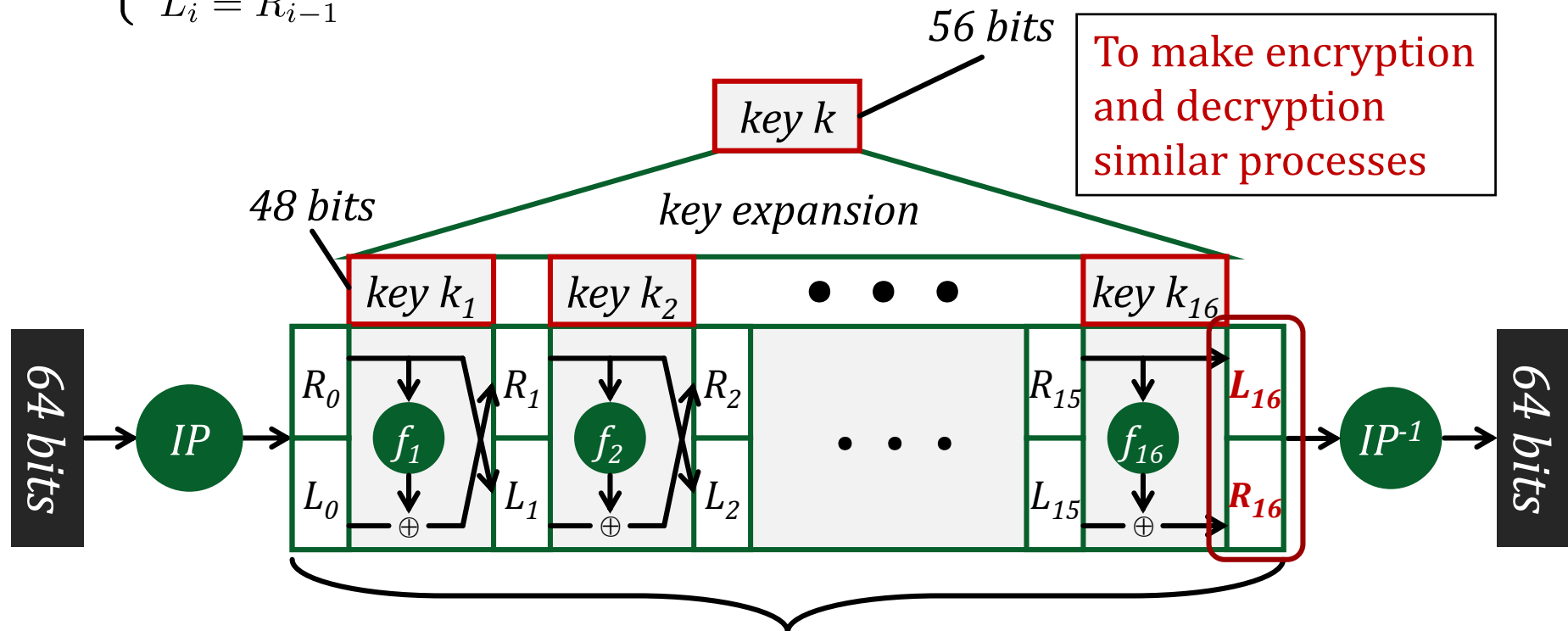


- Inversion is basically the same circuit, with  $f_1, \dots, f_d$  applied *in reverse order*
- General method for building invertible functions (block ciphers) from arbitrary functions.
- Used in many block ciphers ... but **not AES**

# DES: 16 round Feistel network

$$f_1, \dots, f_{16} : \{0, 1\}^{32} \rightarrow \{0, 1\}^{32} \text{ and } f_i(x) = \mathbf{F}(k_i, x)$$

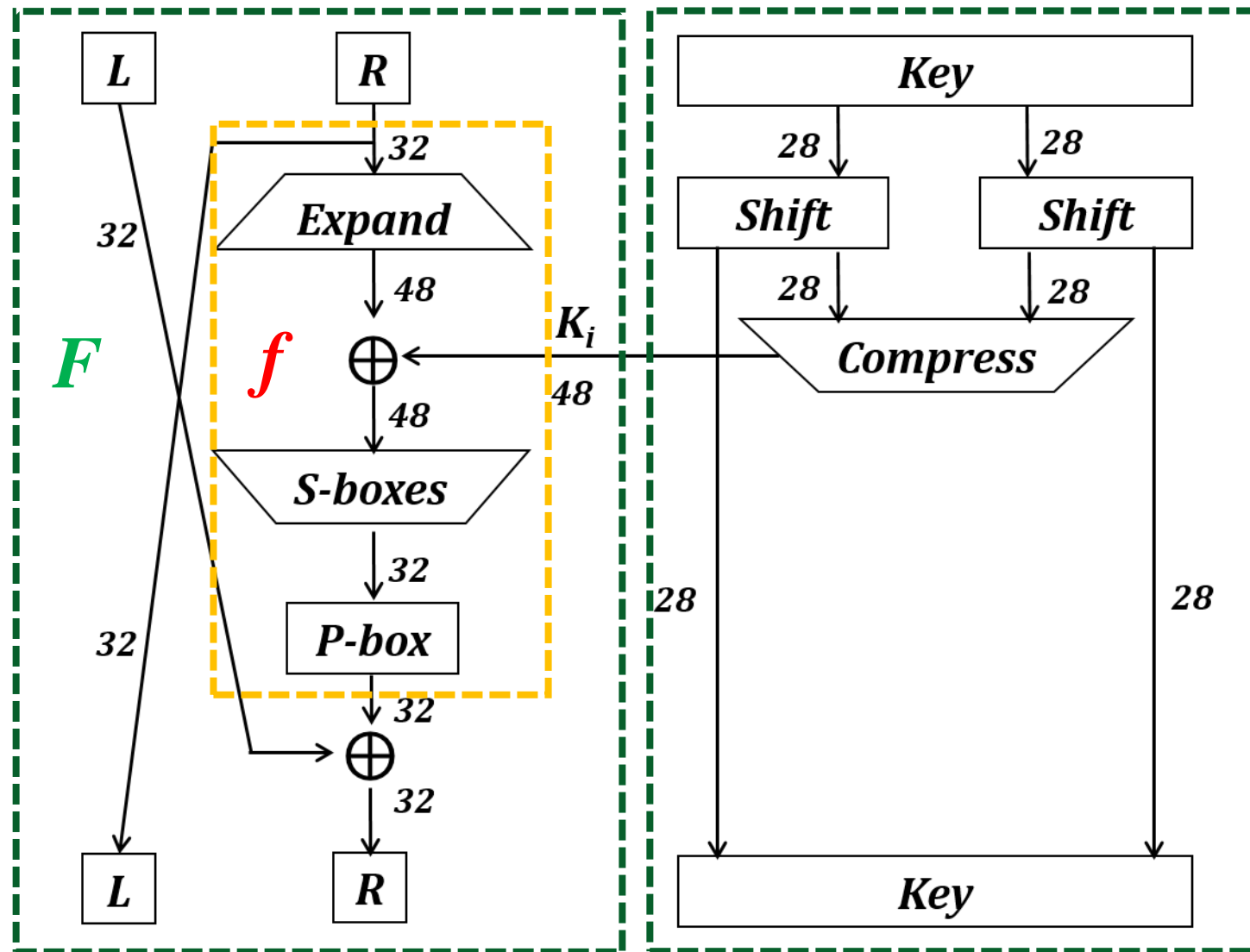
$$\begin{cases} R_i = f_i(R_{i-1}) \oplus L_{i-1} \\ L_i = R_{i-1} \end{cases}$$



16 round Feistel network  
To invert, use keys in reverse order



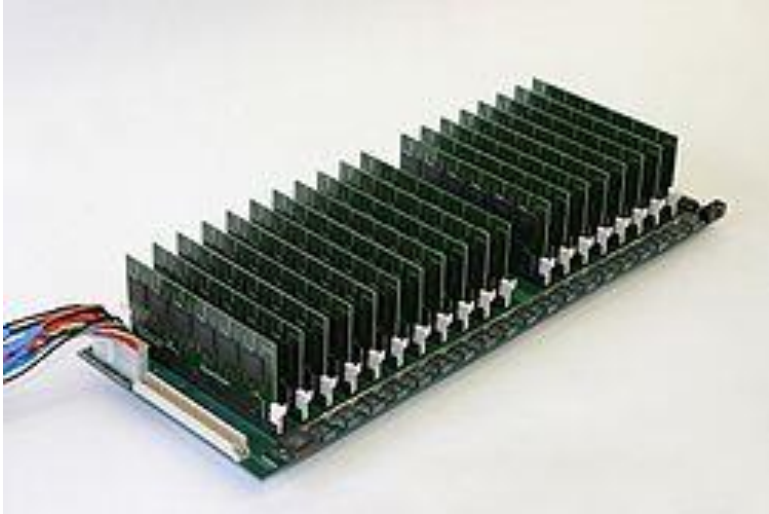
# One Round of DES



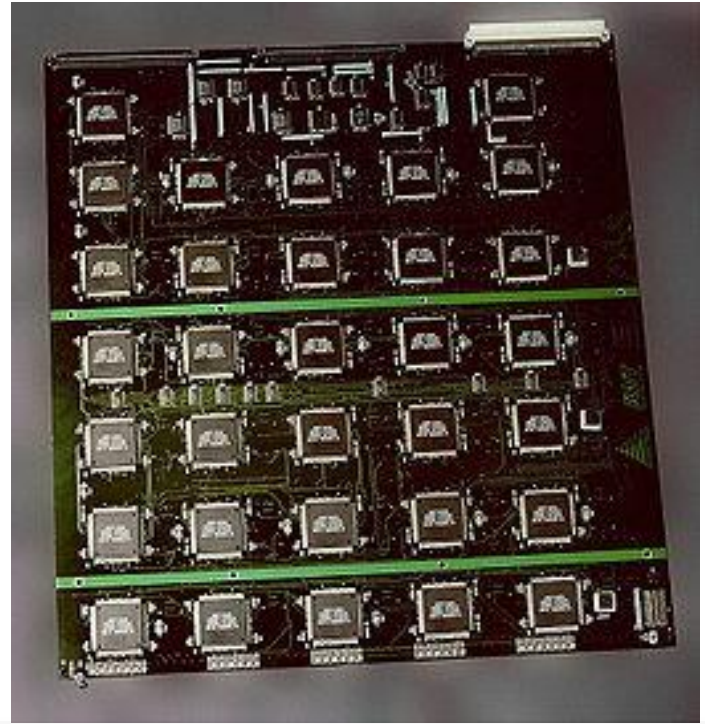
# Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- Thirty+ years of intense analysis has revealed no “back door”
- Attacks, essentially exhaustive key search
- **Inescapable conclusions**
  - Designers of DES knew what they were doing
  - Designers of DES were way ahead of their time

# DES Cracker machine



The COPACOBANA machine, built for US\$10,000 by the Universities of Bochum and *Kiel, Germany*, contains 120 low-cost FPGAs and can perform an exhaustive key search on DES in **6.4 days** on average. The photo shows the backplane of the machine with the FPGAs



The EFF's US\$250,000 *DES cracking machine* contained 1,856 custom chips and could brute force a DES key in a matter of days — the photo shows a DES Cracker circuit board fitted with several Deep Crack chips.

# Secure DES

- Today, 56 bit DES key is too small
  - Exhaustive key search is feasible
- But DES is everywhere, so what to do?
  - Solution: use Multi-encryption to expand the length of the key
    - Make full use of the design of DES
    - Save the investment in software and devices



2-DES?

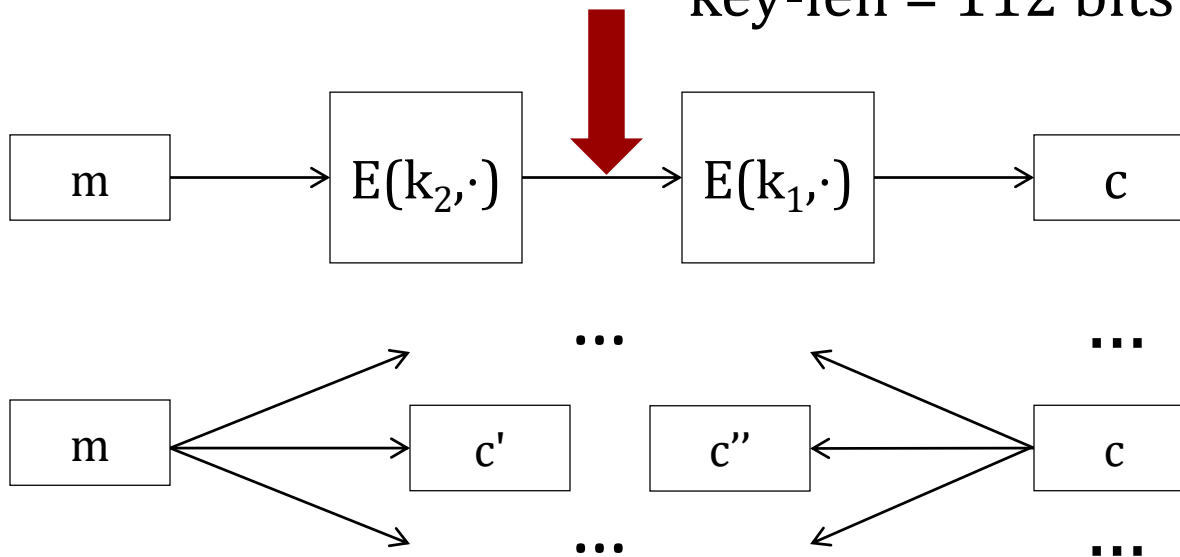
NO!



# Meet in the middle attack

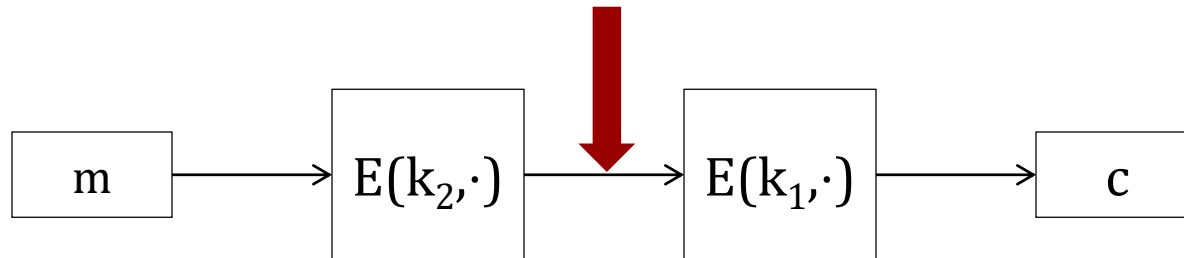
- Define  $2E((k_1, k_2), m) = E(k_1, E(k_2, m))$

key-len = 112 bits for 2DES



- Idea: key found when  $c' = c''$ :  $E(k_i, m) = D(k_j, c)$

# Meet in the middle attack



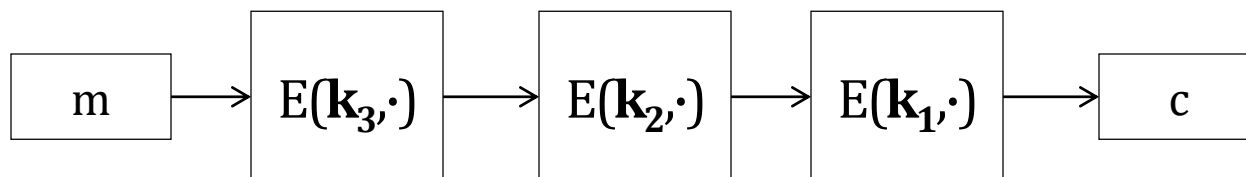
$$\text{Time} = 2^{56} \log(2^{56}) + 2^{56} \log(2^{56}) < 2^{63} \ll 2^{112}$$

[Build & Sort Table]

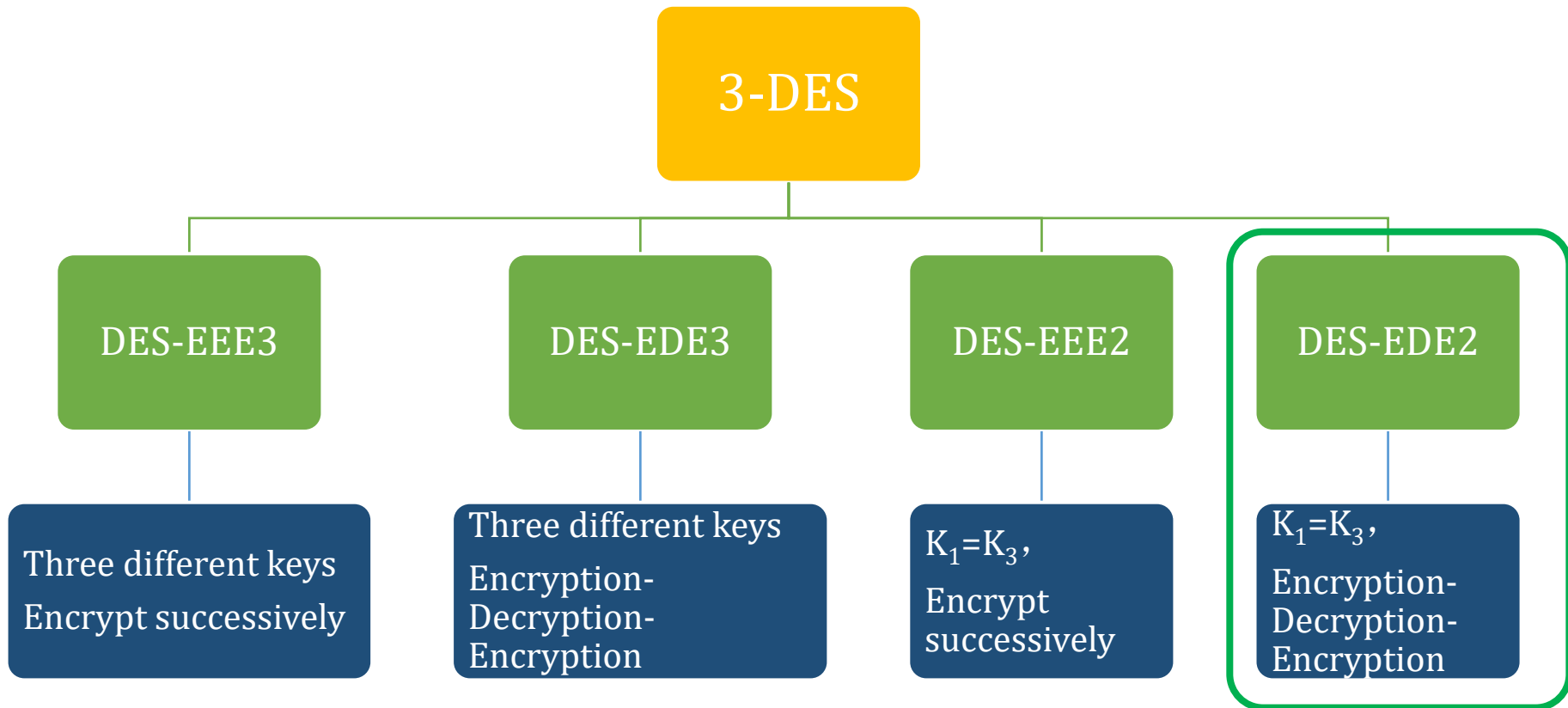
[Search Entries]

$$\text{Space} \approx 2^{56} \text{ [Table Size]}$$

Same attack on 3DES:  $\text{Time} = 2^{118}$ ,  $\text{Space} \approx 2^{56}$



# Triple-DES(3-DES)



# Concerns about DES

- Short key length
  - DES “cracker”, built for \$250K, can break DES in days
  - Distributing the computation makes it faster
- Some (theoretical) attacks have been found
- Non-public design process
- 3-DES is fairly slow



# Advanced Encryption Standard

Brief Description

Algorithm

# AES

Joan Daemen Vincent Rijmen



- Public contest sponsored by NIST in '97
  - Narrowed to 5 finalists
  - 4 years of intense analysis
- Efficiency and security taken into account
- 128-bit key length and 128-bit block size (minimum)
- Rijndael selected as the AES
  - Supports variety of block/key sizes

# AES Candidates (15->5):

*Metrics: The same security performance as 3-DES and faster...*

- **MARS**

- submitted by International Business Machines Corporation (U.S.)

- **RC6**

- submitted by RSA Laboratories (U.S.)

- **Rijndael**

- submitted by Joan Daemen and Vincent Rijmen (Belgium)

- **Serpent**

- submitted by Ross Anderson (U.K.), Eli Biham (Israel), and Lars Knudsen (Norway)

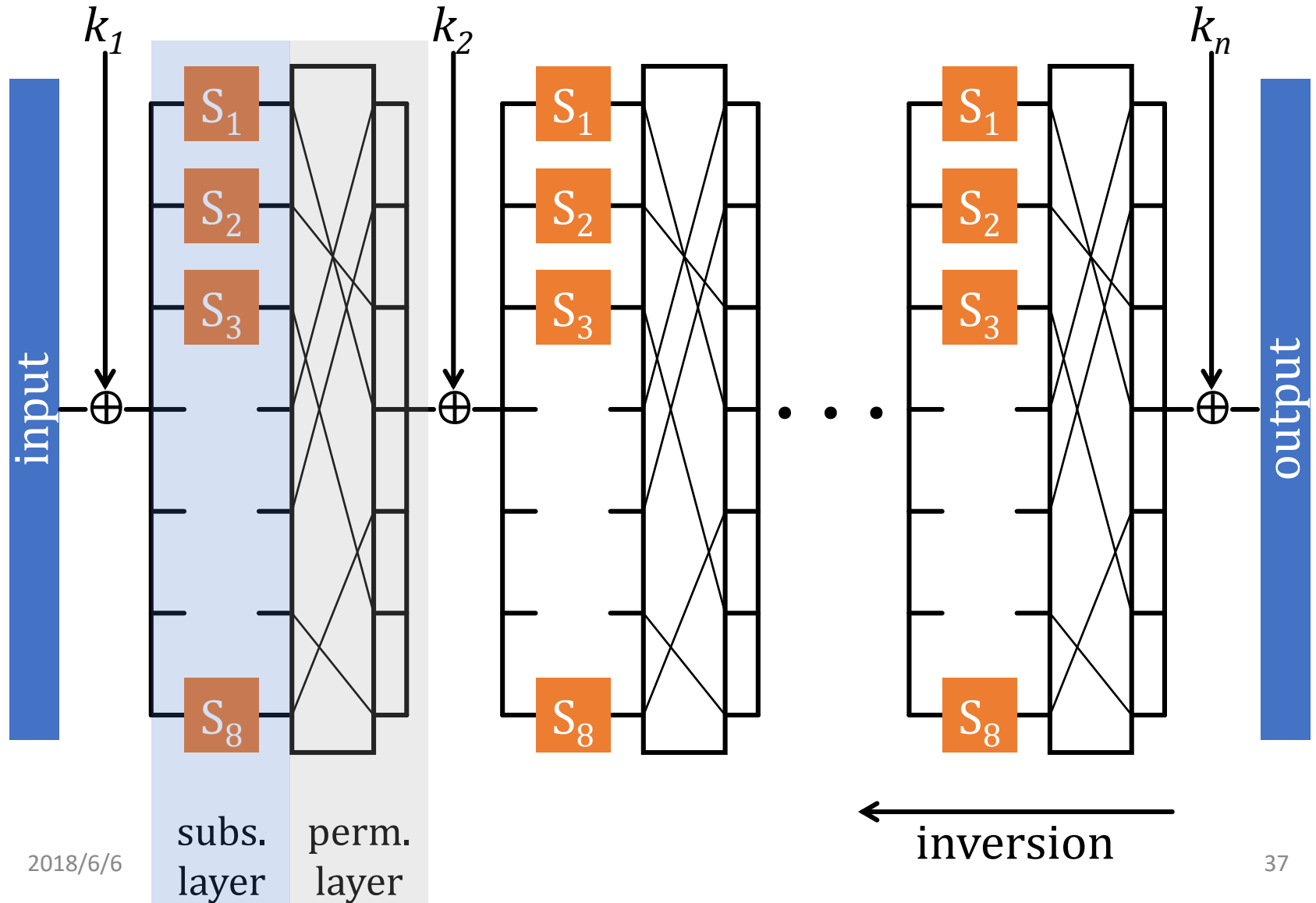
- **Twofish**

- submitted by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson (U.S.)

# Advanced Encryption Standard

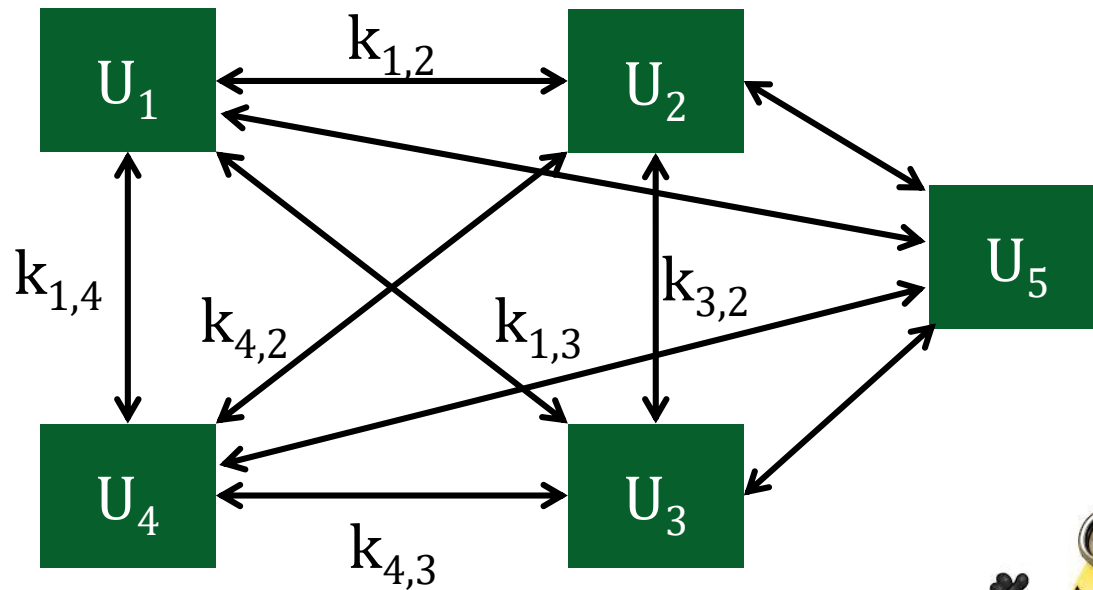
- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected
    - Pronounced like “Rain Doll” or “Rhine Doll”
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)
  - AES is based on the idea of **substitution-permutation** networks

# AES: Subs-Perm network



# Key management

Problem: Communicating among  $n$  users.



Total:  $O(n^2)$  keys

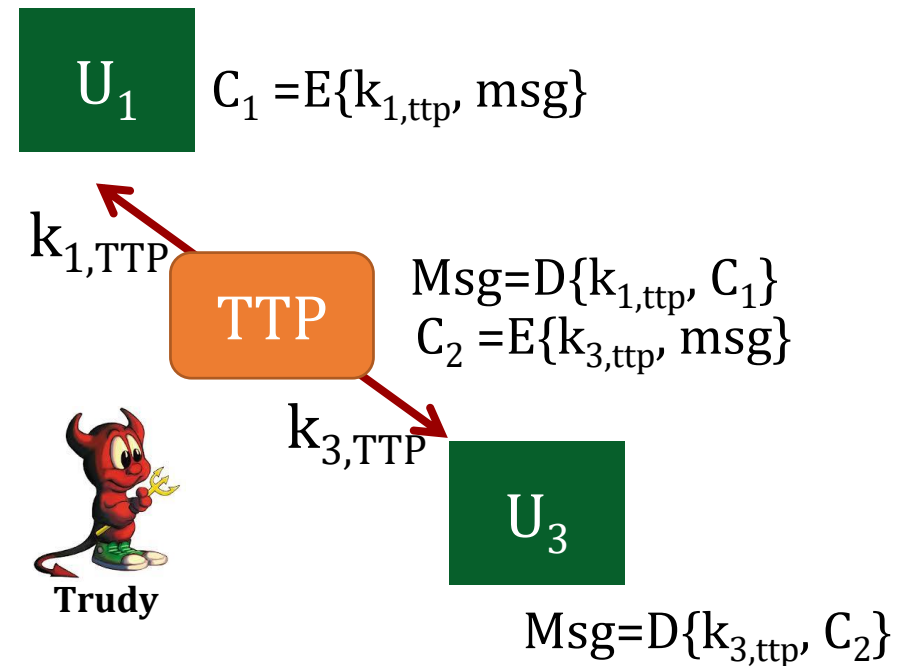
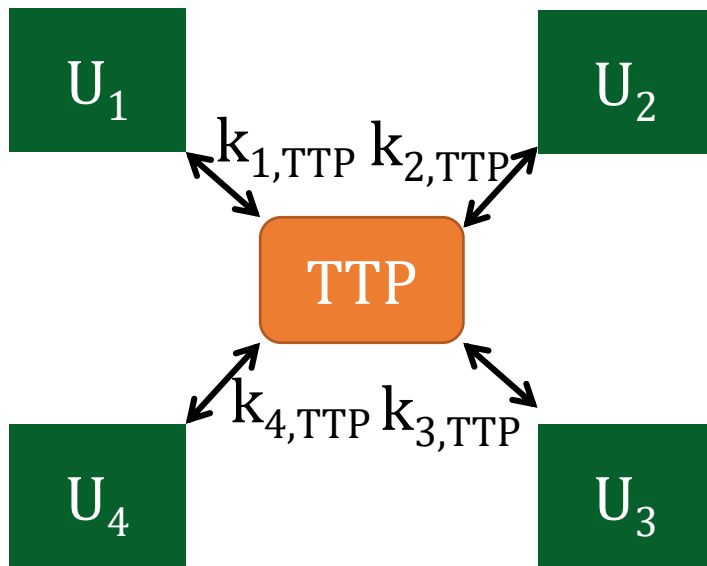
How to Fix this?



# One Solution: Trusted Third Party (TTP)



Everyone needs only one key



Can we remove the TTP as a communication and privacy bottleneck?



# Key question

Can we generate shared keys without an online trusted 3rd party?



Answer: yes!

Starting point of public-key cryptography:

- Merkle (1974), Diffie-Hellman (1976), RSA (1977)

Broken

Key Agreement

Encryption  
Digital Signature  
Key Agreement

- More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)



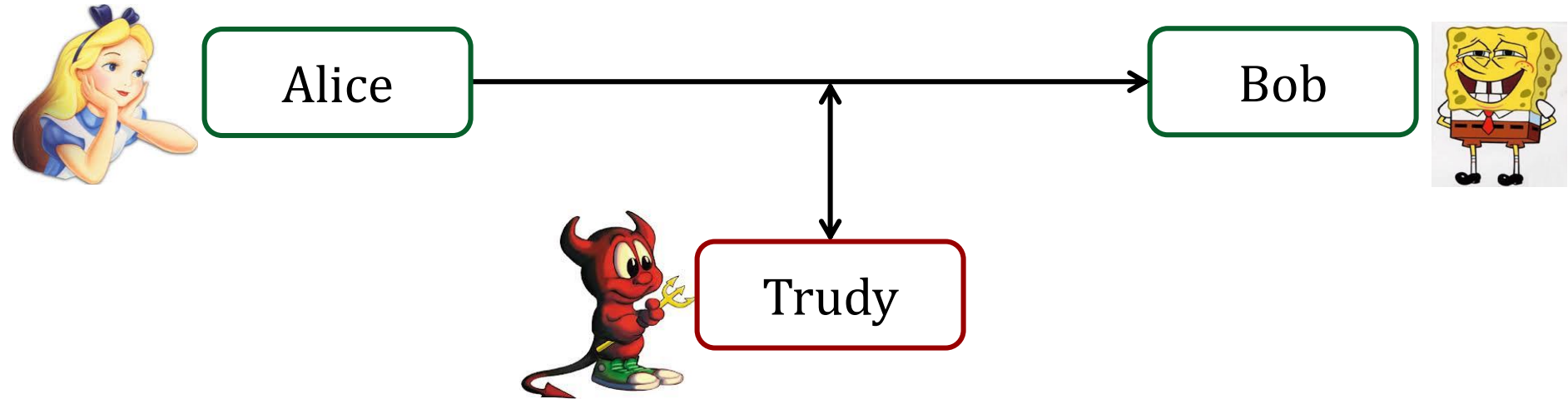


Whitfield Diffie



Martin Hellman

# The Diffie-Hellman Protocol



**Goal**: establish shared key for security against **eavesdroppers** without a TTP

# Modulo

- The **modulo** operation finds the remainder after division of one number by another.
- Given two positive numbers,  $a$  (the dividend) and  $n$  (the divisor),  $a$  modulo  $n$  (abbreviated as  $a \bmod n$ ) is the remainder of the Euclidean division of  $a$  by  $n$ .
- Example.
  - $5 \bmod 2 = ?$

# Discrete Log

- **Logarithms** are the inverse of **exponentiation**.

$$b^y = x \text{ is equivalent to } \log_b(x) = y$$

Consider arithmetic **mod**  $p$ , where  $p$  is a prime. The discrete log to the base  $b$  of  $x$  is an integer  $y$  such that  $b^y \bmod p = x$ .

Example. Let  $p = 17$ . Then:

$$3^4 \bmod 17 = 81 \bmod 17 = 13. \text{ So } 3^4 = 13 \pmod{p}$$

Then the discrete  $\log_3(13) = ?$

# “Discrete Log” Problem

**Easy**: Given  $b$ ,  $y$ , and  $p$ , compute  $x = b^y \bmod p$

- See “Handbook of Applied Cryptography”, available free online

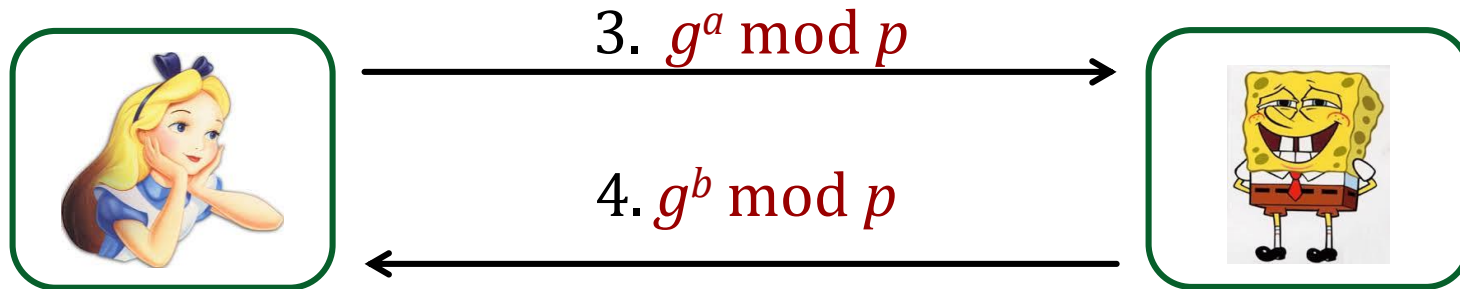
**Believed Hard**: Given  $b$ ,  $p$ ,  $x$ , compute  $y$  such that  $b^y \bmod p = x$ .

# Key Exchange with Discrete Log

Setup: Fix a public large prime  $p$  ( $\sim 600$  digits  $\approx 2048$  bits) and a public number  $g$  between 0 and  $p$ .

1. Select  $a$  from  $[0, p-1)$

2. Select  $b$  from  $[0, p-1)$



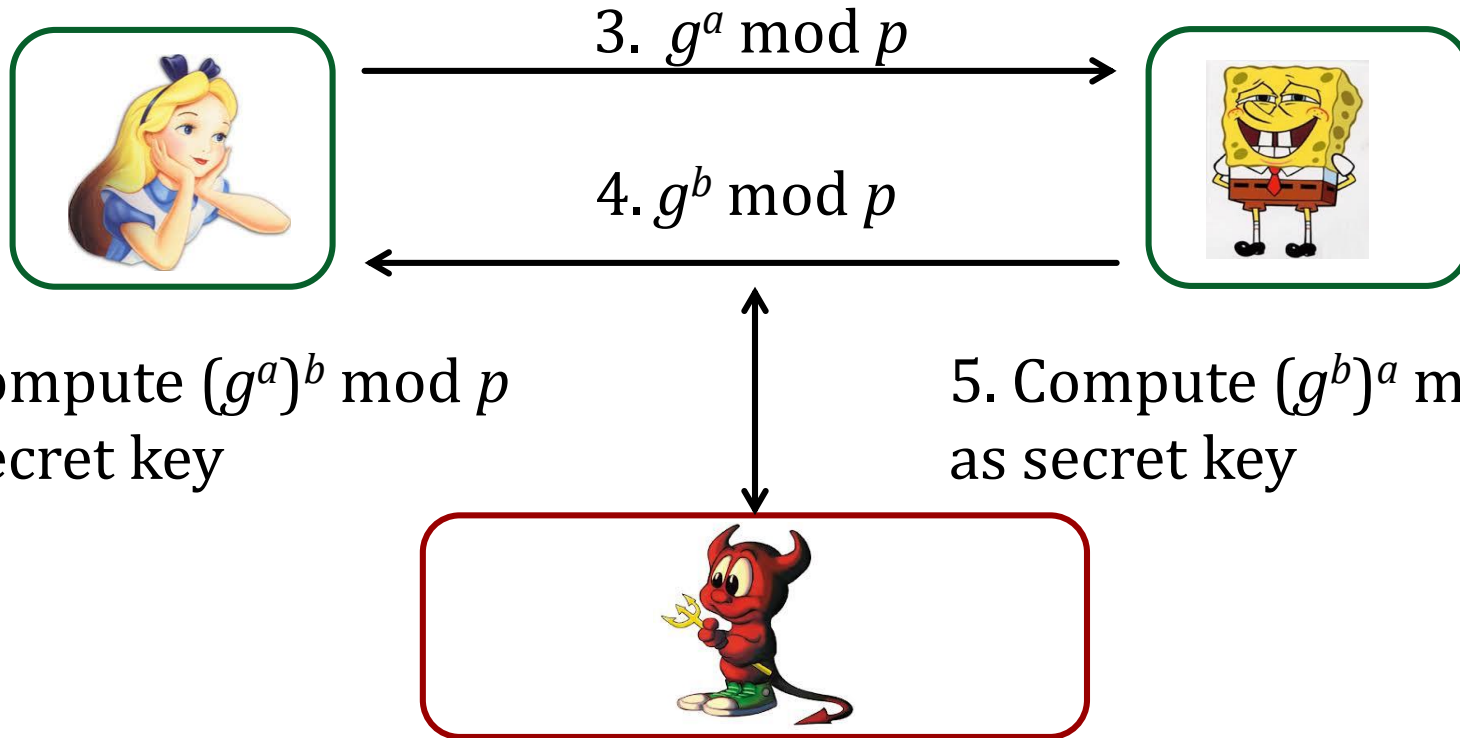
5. Compute  
 $k = (g^b)^a \bmod p$

5. Compute  
 $k = (g^a)^b \bmod p$

6. Use  $k$  for symmetric (authenticated) encryption.

1. Select  $a$  from  $[0, p-1)$

2. Select  $b$  from  $[0, p-1)$



Trudy observes:  $g, g^a, g^b$

**Goal:** compute  $a$  (or  $b$ ) (i.e., calculate the discrete log)  
or compute  $g^{ab}$

# How hard is the *DH* function mod $p$ ?

Suppose prime  $p$  is  $n$ -bits long.

Best known algorithm \*:

$$\exp \left( \left( \sqrt[3]{\frac{64}{9}} + o(1) \right) (\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}} \right) \approx e^{\tilde{O}(n^{1/3})}$$

Any other attack  
available?

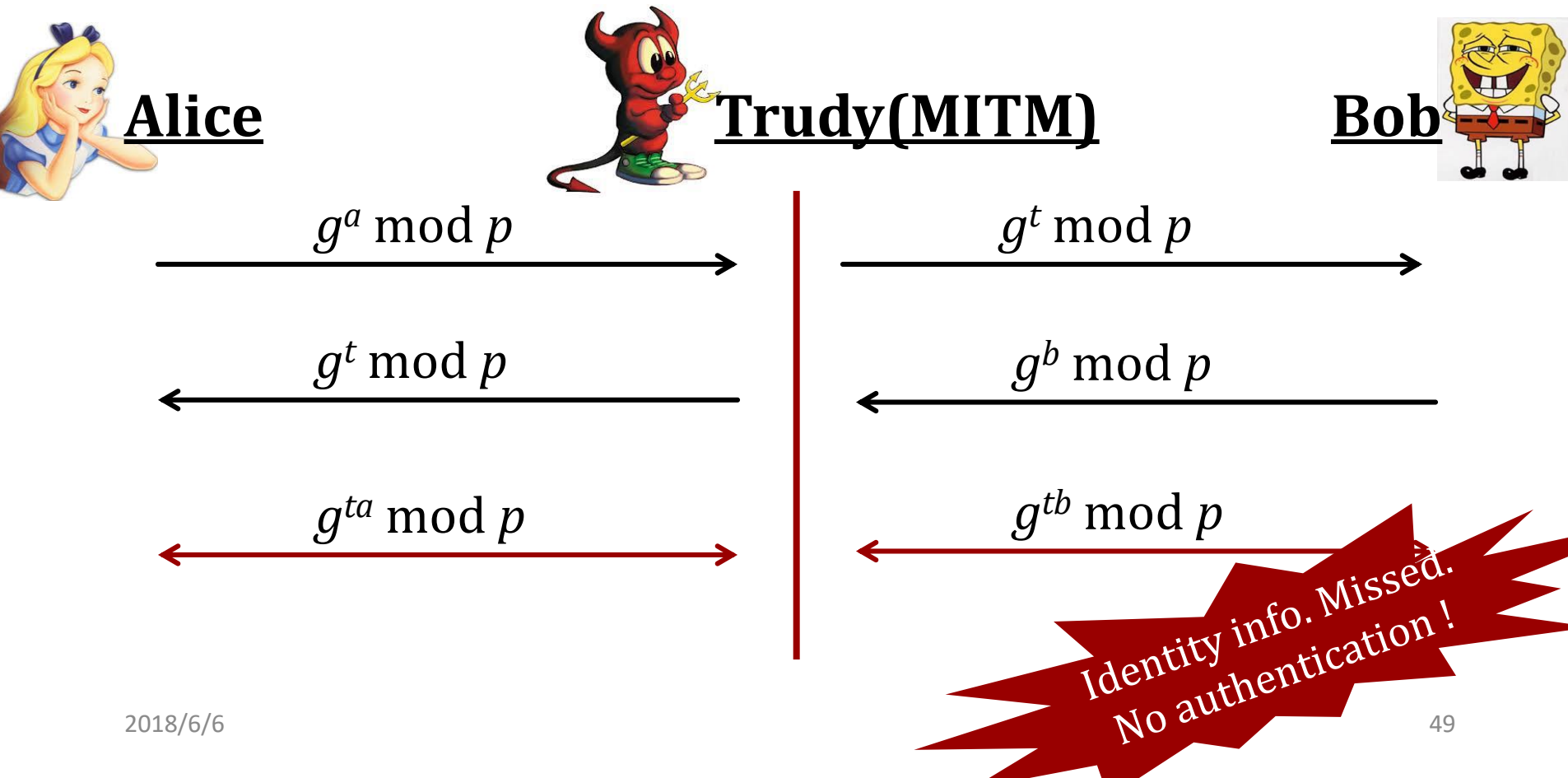


\*  $\tilde{O}$ -hat means left lots of lower-order terms off



# MITM Adversary

Diffie-Hellman is insecure against active “Man In The Middle (MITM)” attacks



# Public Key Crypto System

## Overview

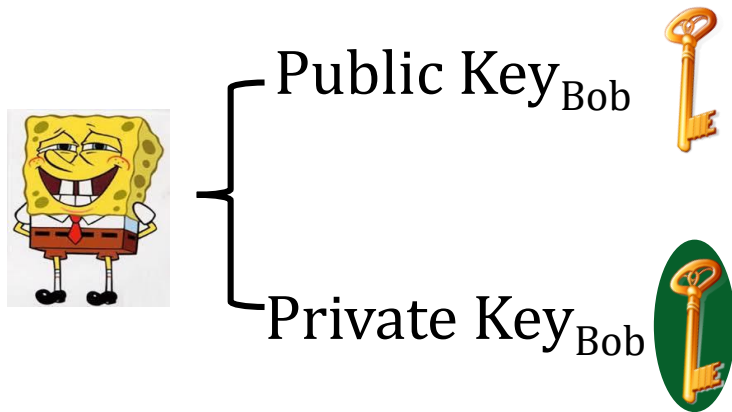
Public Key System for Encryption (Confidentiality)

Public Key System for Authentication (Source Integrity)

# Public Key Cryptography

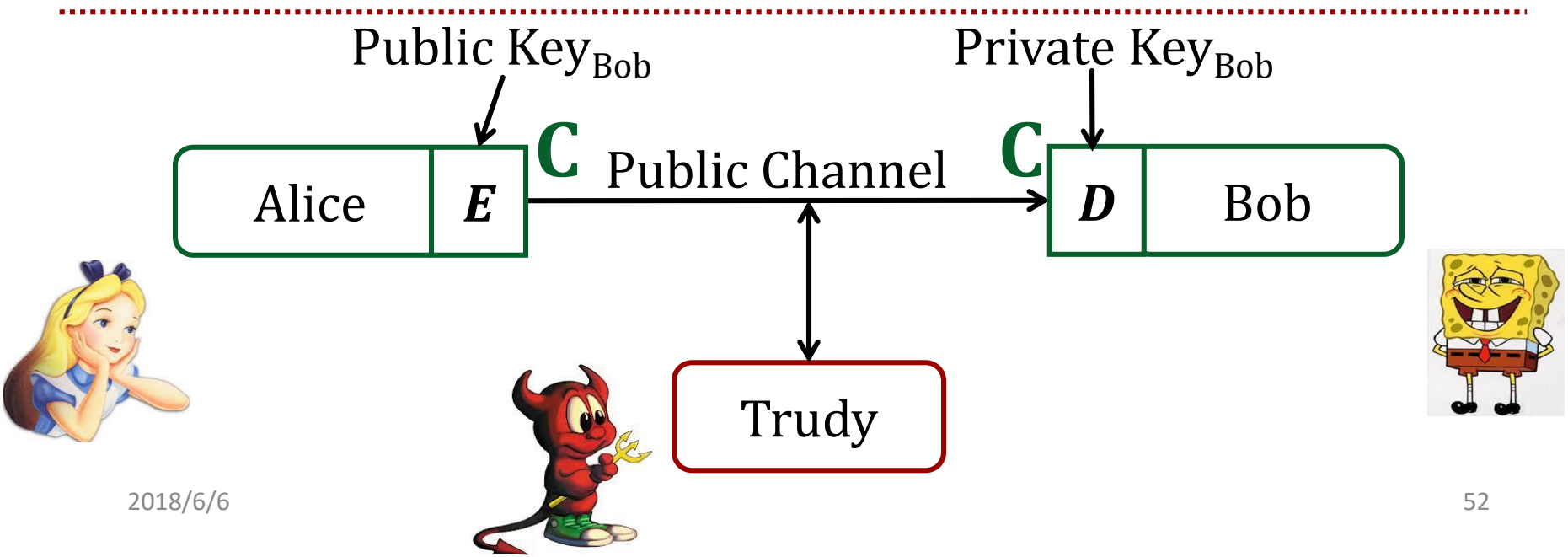
- Encryption
  - Suppose we **encrypt**  $M$  with Bob's *public key*
  - Bob's *private key* can **decrypt** to recover  $M$
- Digital Signature
  - **Sign** by “encrypting” with your *private key*
  - Anyone can **verify** signature by “decrypting” with *public key*
  - But only you could have signed
  - Like a handwritten signature, but way better...

# Public Key Crypto--Encryption

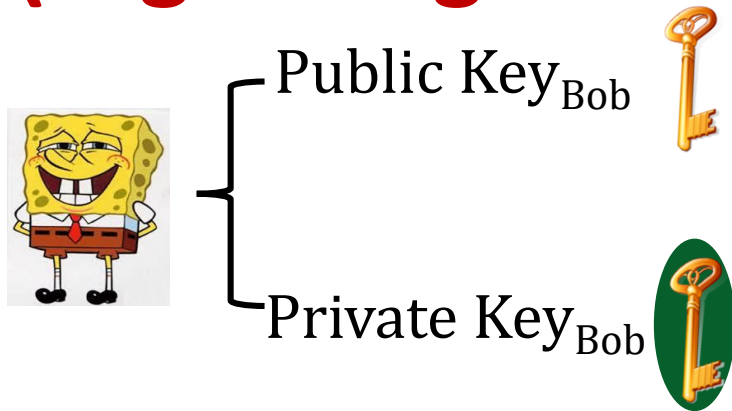


- Everyone can send Alice an **encrypted** msg.
- Only Alice can **decrypt** and read the msg.

Confidentiality

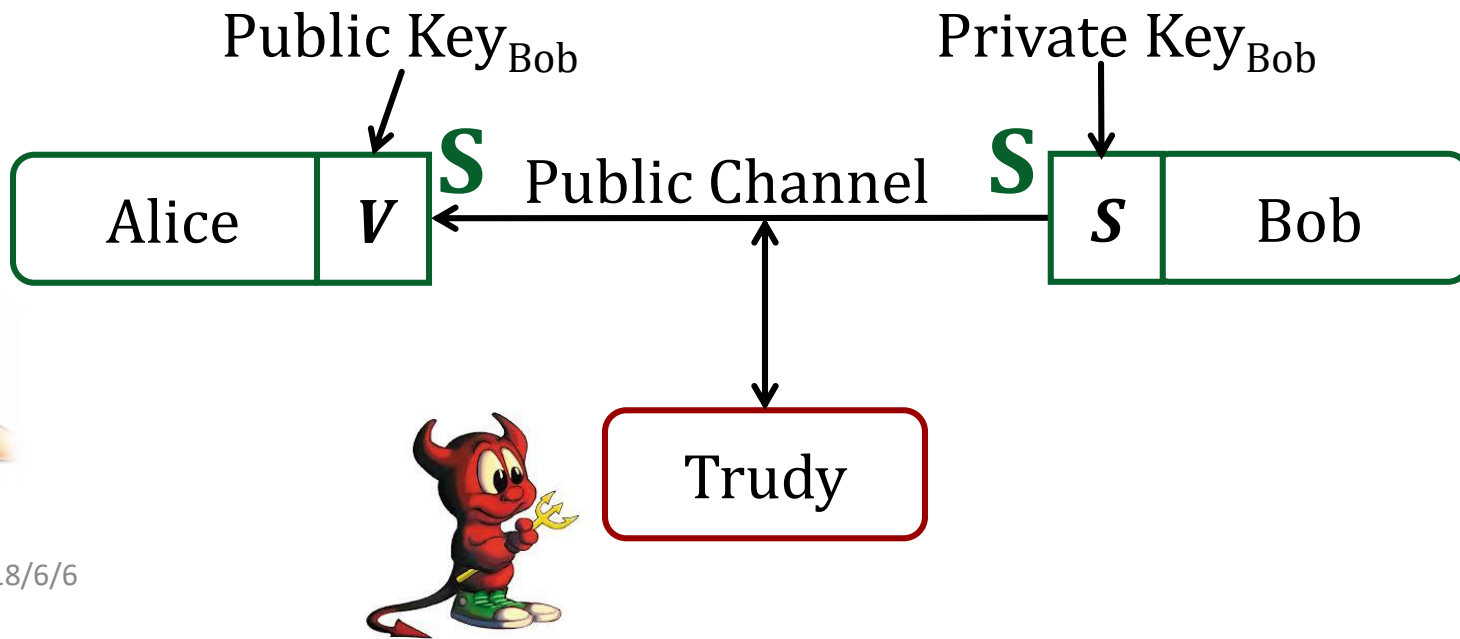


# Public Key Crypto—Authentication (Digital Signature)



- Only Bob can Encrypt and **Sign** the msg.
- Everyone can **Verify** Bob's Signature of the msg.

Source Integrity



# RSA

- The algorithm was published in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT.
- RSA is a block cipher .
  - The plaintext and ciphertext are integers between  $0$  and  $n-1$  for some  $n$ .
- A typical size for  $n$  is  $1024$  bits, or 309 decimal digits.
  - $n$  is no less than  $2^{1024}$ .



*Ron Rivest, Adi Shamir, Len Adleman*

Source: from Wikipedia

# Description of the algorithm

RSA is an exponentiation cipher.

- Choose two large prime numbers  $p$  and  $q$ , and let  $n=pq$ .
- **The totient  $\varphi(n)$**  of  $n$  is the number of numbers less than  $n$  with no factors in common with  $n$ .
- Choose an integer  $e < n$  that is relatively prime to  $\varphi(n)$ .
- Find a second integer  $d$  such that  $ed \bmod \varphi(n) = 1$ .
- The public key is  $(e, n)$ , and the private key is  $d$ .
- Let  $m$  be a message. Then:
  - $c = m^e \bmod n$  (encryption)     $m = c^d \bmod n$  (decryption)

## Example: $\varphi(n)$

- Let  $n=10$ . Then numbers that are less than 10 and are relatively prime to  $n$  are 1,3,7 and 9. Hence  $\varphi(10)=4$ .
- Similarly, if  $n=21$ , the numbers that are relatively prime to  $n$  are 1,2,4,5,8,10,11,13,16,17,19, and 20. So  $\varphi(21)=12$ .



# RSA algorithm

## Key Generation

Select $p, q$	$p$ and $q$ both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\varphi(n) = (p-1)(q-1)$	
Select integer $e$	$\gcd(\varphi(n), e) = 1; 1 < e < \varphi(n)$
Calculate $d$	$d \equiv e^{-1} \pmod{\varphi(n)}$
Public key	$Pk = \{e, n\}$
Private key	$Sk = \{d\}$

## Encryption

Plaintext	$M < n$
Ciphertext	$C \equiv M^e \pmod{n}$

## Decryption

Ciphertext	$C$
Plaintext	$M \equiv C^d \pmod{n}$

# RSA algorithm

## Key Generation

Select $p, q$	$p$ and $q$ both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer $e$	$\gcd(\phi(n), e) = 1: 1 < e < \phi(n)$
Calculate $d$	
Public key	
Private key	

- $e$  and  $N$  are public
- If Trudy can factor  $N = pq$ , she can use  $e$  to easily find  $d$  since  $ed = 1 \bmod (p-1)(q-1)$
- **Factoring the modulus breaks RSA**

## Encryption

Plaintext	$M < n$
Ciphertext	$C \equiv M^e \bmod n$

## Decryption

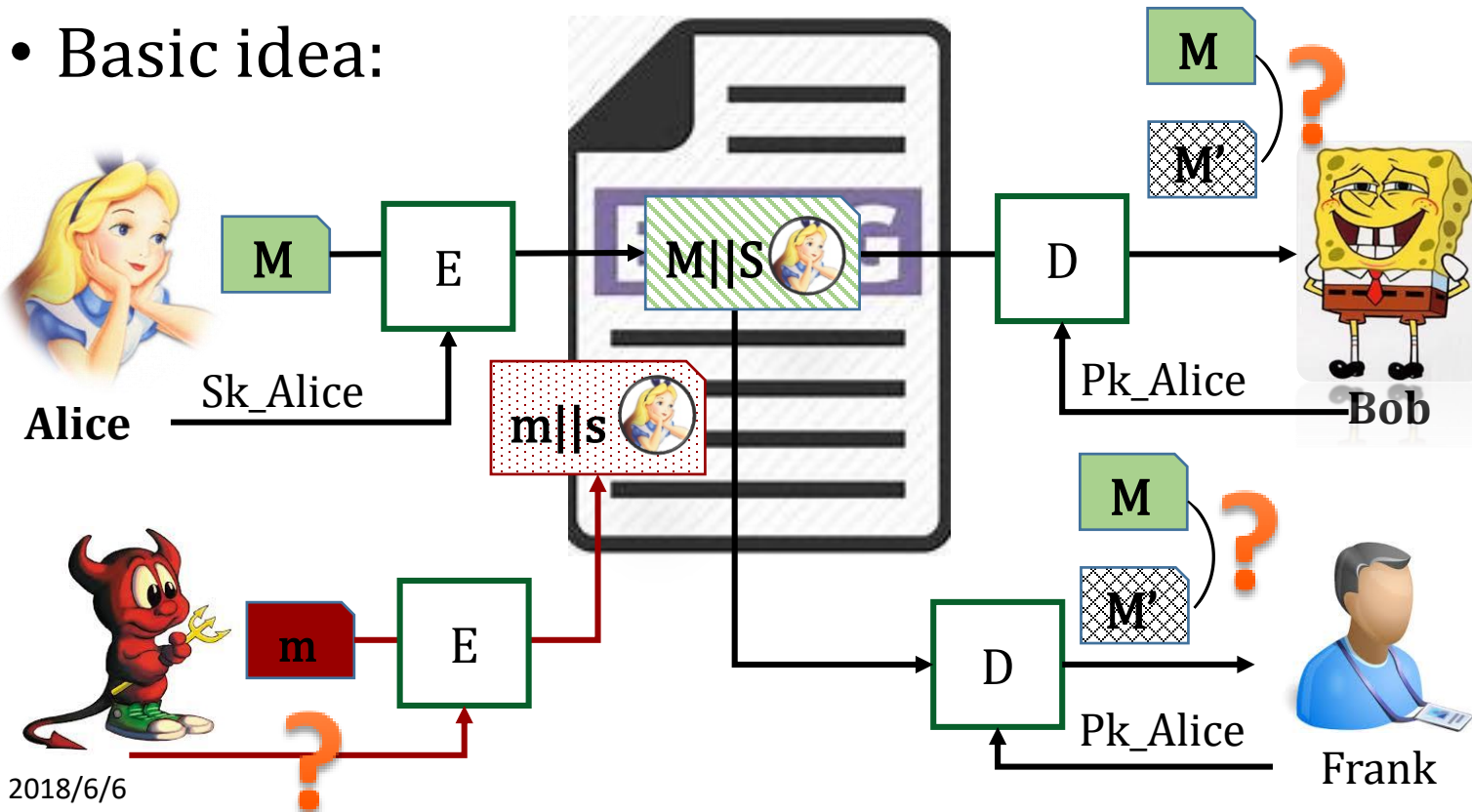
Ciphertext	$C$
Plaintext	$M \equiv C^d \bmod n$

# Example: *Secrecy*

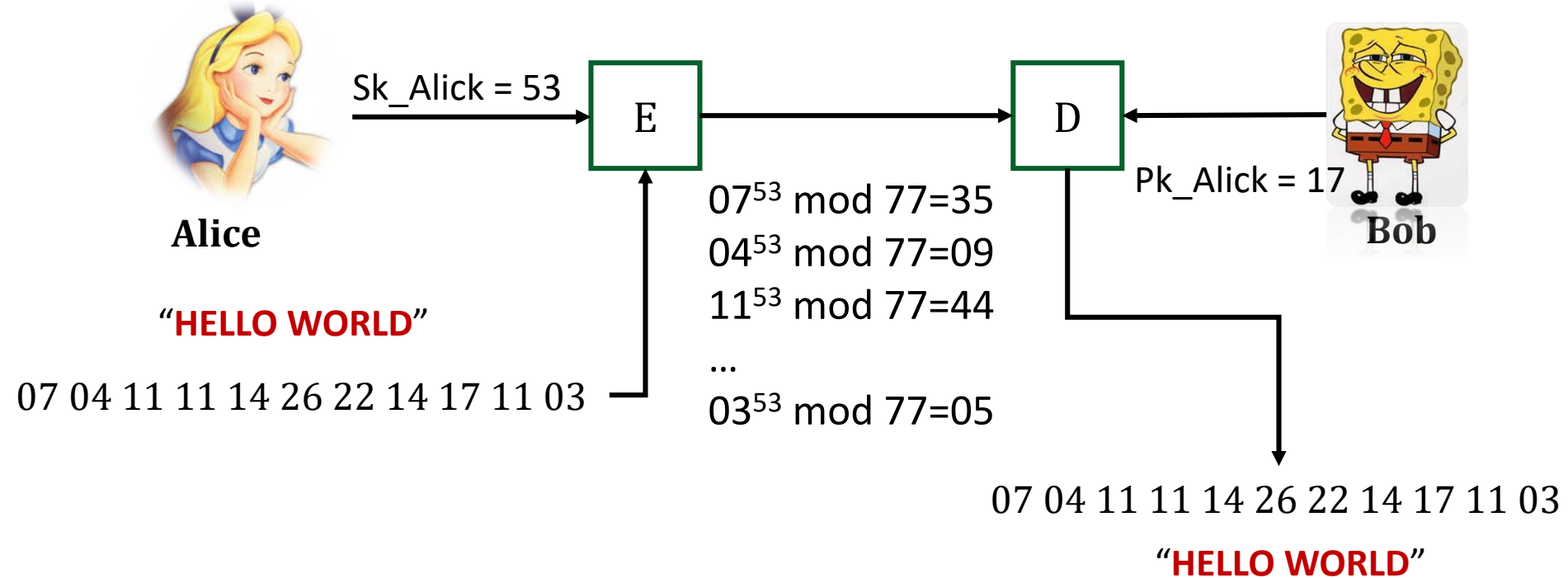
- Let  $p=7$  and  $q=11$ . Then  $n=77$  and  $\varphi(n)=60$ .
- **Alice** chooses  $e=17$ , so her private key is  $d=53$ .
- Each plaintext character is represented by a number between 00(A) and 25(Z); 26 represents a blank.
- **Bob** wants to send Alice the message
  - “**HELLO WORLD**”
- Using Alice’s public key, the ciphertext is
  - $07^{17} \bmod 77=28$
  - $04^{17} \bmod 77=16$
  - $11^{17} \bmod 77=44$
  - ...
  - $03^{17} \bmod 77=75$
  - Obtain 28 16 44 44 42 38 22 42 19 44 75

# Authentication

- In addition to confidentiality, RSA can provide data and origin authentication.
- Basic idea:



# Example: Authentication



# Uses for Public Key Crypto

# Uses for Public Key Crypto

- Key agreement
- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Authentication
- Digital signature provides integrity and **non-repudiation**
  - No non-repudiation with symmetric keys

# Public Key Infrastructure

Public Key Certificate

Certificate Authority

PKI Trust Models



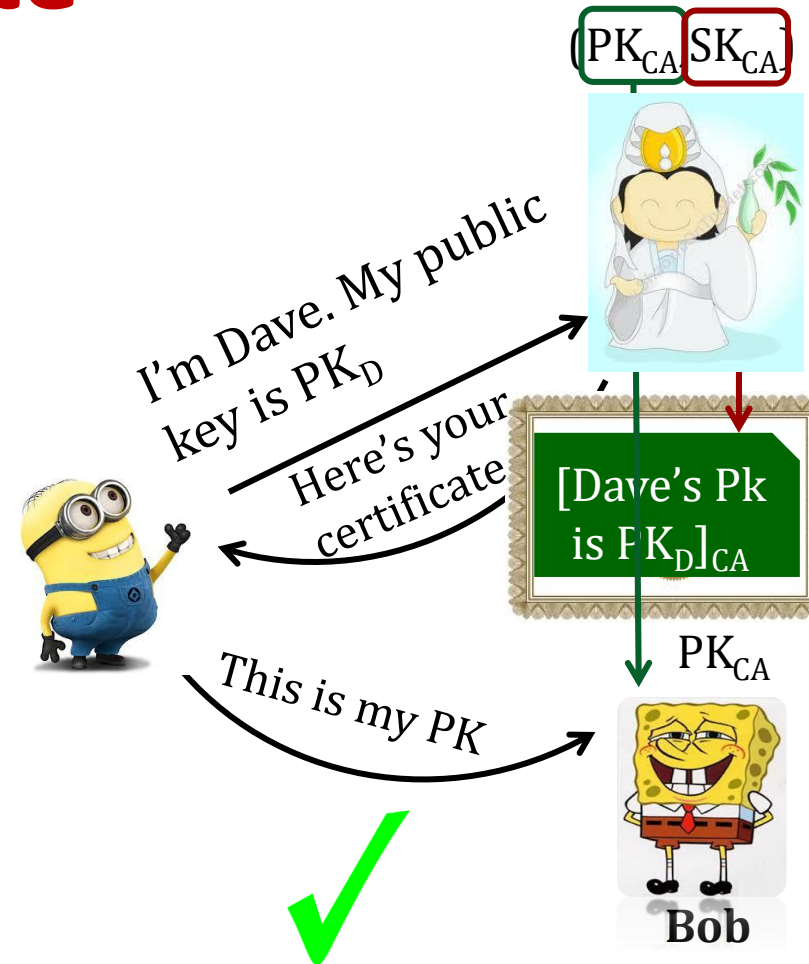
# How to trust the public key?



- The need for verifying identity to establish trust
  - Can/Do we trust the public key we received?
- Public key distribution
  - *Public announcement*: users distribute public keys to recipients or broadcast to community at large
  - *Publicly available directory*: can obtain greater security by registering keys with a public directory

# Public Key Certificate

- A **certificate** binds a *public key* and user's *identity*
- A **certificate authority** (CA) signs the certificate
- R1: Certificates can be verified by anyone who knows the CA's public key
- R2: Certificates allow key exchange without real-time access to CA



$\{M\}_K$ : Encrypt  $M$  with  $K$ 's public key  
 $[M]_K$ : Sign  $M$  with  $K$ 's private key

# Public Key Certificate (Basic)



- **Certificate** contains name of user and user public key (and possibly other info)
- It is *signed* by the issuer, a *Certificate Authority*(CA), such as VeriSign

$M = (\text{Alice}, \text{Alice's public key}), S = [M]_{CA}$

**Alice's Certificate** =  $(M, S)$

- Signature on certificate is verified using CA's public key:

Verify that  $M = \{S\}_{CA}$

# Certificate Authority

- Certificate authority (CA) is a trusted 3rd party (TTP) — **creates** and **signs** certificates
- Verify signature to verify integrity & identity of **owner of corresponding private key**
  - Does **not** verify the identity of the **sender** of certificate — certificates are public keys!
- A common format for certificates is X.509

# X.509 Certificate Format

Version	Version v1	Version v2	Version v3
Serial number			
Signature algorithm identifier			
Publisher name			
Validity period			
Entity name			
Information about the entity public key			
Unique publisher ID		Version v2	
Unique entity ID			
Additions			
Signature	All versions		

Amazon.com: Online Shopping x Settings x Check if a site's connection is x Se

Secure https://www.amazon.com

NEW & INTERESTING

amazon Try Prime

Departments Amazon@CollegePark

您是在 Amazon

OK

**CA Hierarchy**

- VeriSign Class 3 Public Primary Certification Authority - G5
  - Symantec Class 3 Secure Server CA - G4
    - www.amazon.com

**www.amazon.com**

Issued by: Symantec Class 3 Secure Server CA - G4

Expires: Monday, 1 January 2018 at 7:59:59 AM China

✓ This certificate is valid

▼ Details

Subject Name	
Country	US
State/Province	Washington
Locality	Seattle
Organization	Amazon.com, Inc.
Common Name	www.amazon.com
Issuer Name	

# CAs and Trust

- Certificates are trusted if signature of CA verifies
- Chain of CA's can be formed, head CA is called root CA
- In order to verify the signature, the public key of the root CA should be obtain.
- TRUST is centralized (to root CA's) and hierarchical——No standard
  - We mention 3 generic “trust models”

# PKI Trust Models

- Monopoly model
  - One universally trusted organization is the CA for the known universe
  - Big problems if CA is ever compromised
  - Who will act as CA???
    - System is useless if you don't trust the CA!



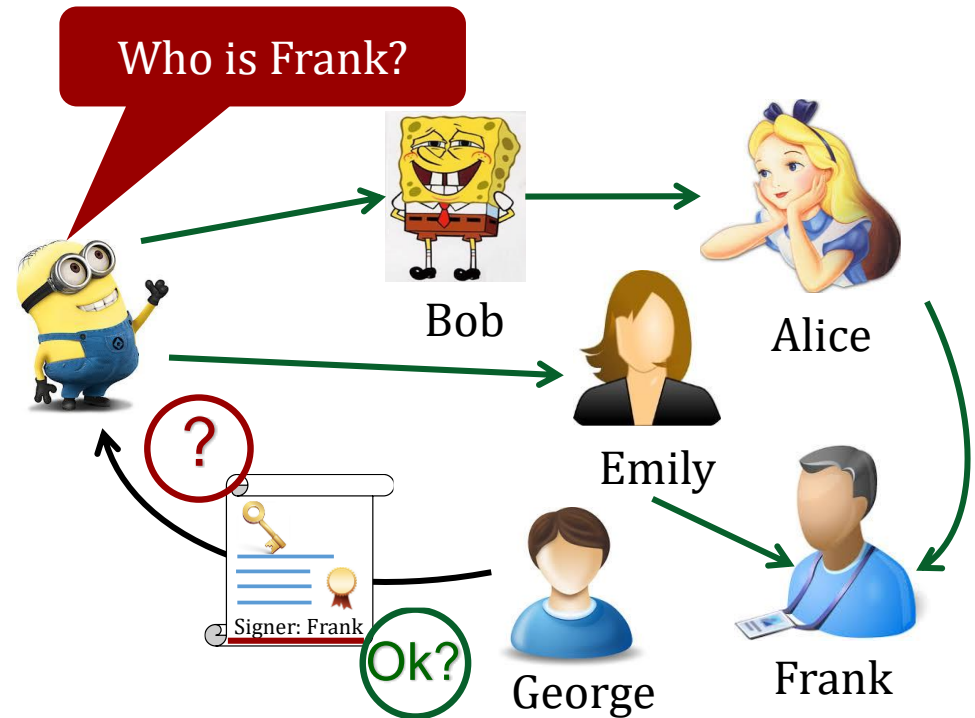


# PKI Trust Models

- Oligarchy
  - Multiple trusted CAs
  - This is approach used in browsers today
  - Many roots nodes in the Internet
    - As a result, many root certification authorities
    - VeriSign, Entrust, and etc.
    - See them in your browser
  - Browser may have 80 or more certificates, just to verify certificates!
  - User can decide which CAs to trust

# PKI Trust Models

- Anarchy model
  - Everyone is a CA...
  - Users must decide who to trust
  - This approach used in PGP: “Web of trust”
- Why is it anarchy?
  - Example ...



- **Many** other trust models and PKI issues

# Confidentiality in the Real World

Symmetric Key vs. Public Key

Real World Confidentiality

# Symmetric Key vs Public Key

- Symmetric key +’s
  - **Speed**
  - No public key infrastructure (PKI) needed
- Public Key +’s
  - **Signatures** (non-repudiation)
  - No *shared* secret (but, private keys...)

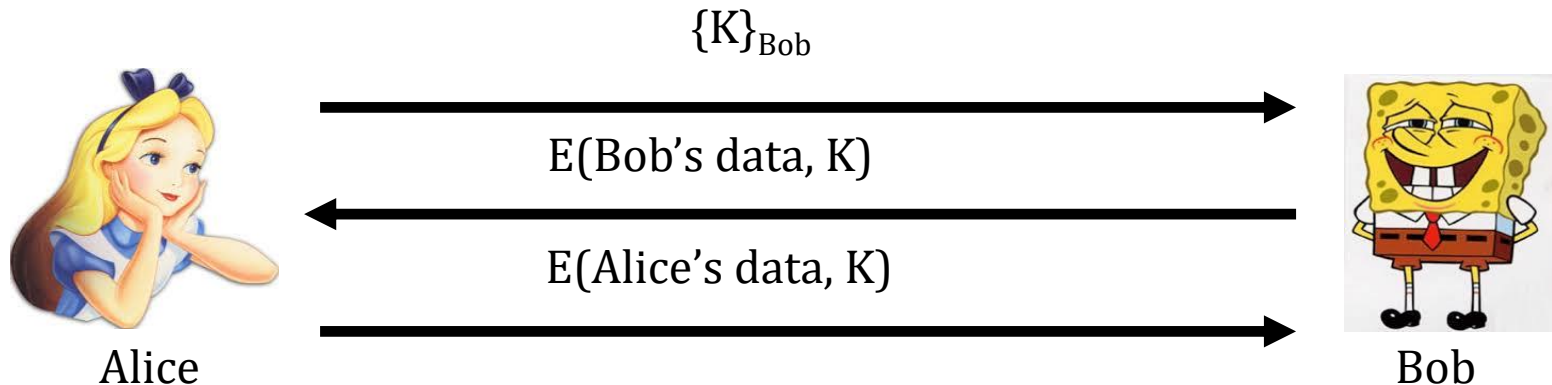
# Notation Reminder

- Public key notation
  - Sign M with Alice's **private key**  
 $[M]_{\text{Alice}}$
  - Encrypt M with Alice's **public key**  
 $\{M\}_{\text{Alice}}$
- Symmetric key notation
  - Encrypt P with symmetric key K  
 $C = E(P, K)$
  - Decrypt C with symmetric key K  
 $P = D(C, K)$

# Real World Confidentiality

- **Hybrid cryptosystem**

- Public key crypto to establish a key
- Symmetric key crypto to encrypt data...



# Lecture Plan

- Basis of Cryptography
  - Concepts
  - Symmetric Key Crypto System
  - Public Key Crypto System
- **Applications of Crypto Systems**
  - Hash Function
  - Password-based Authentication
  - Secure Socket Layer (SSL)
- Labs

# Hash Function

Motivation (Data Integrity)

Definition & Characteristics

Popular Crypto Hashes

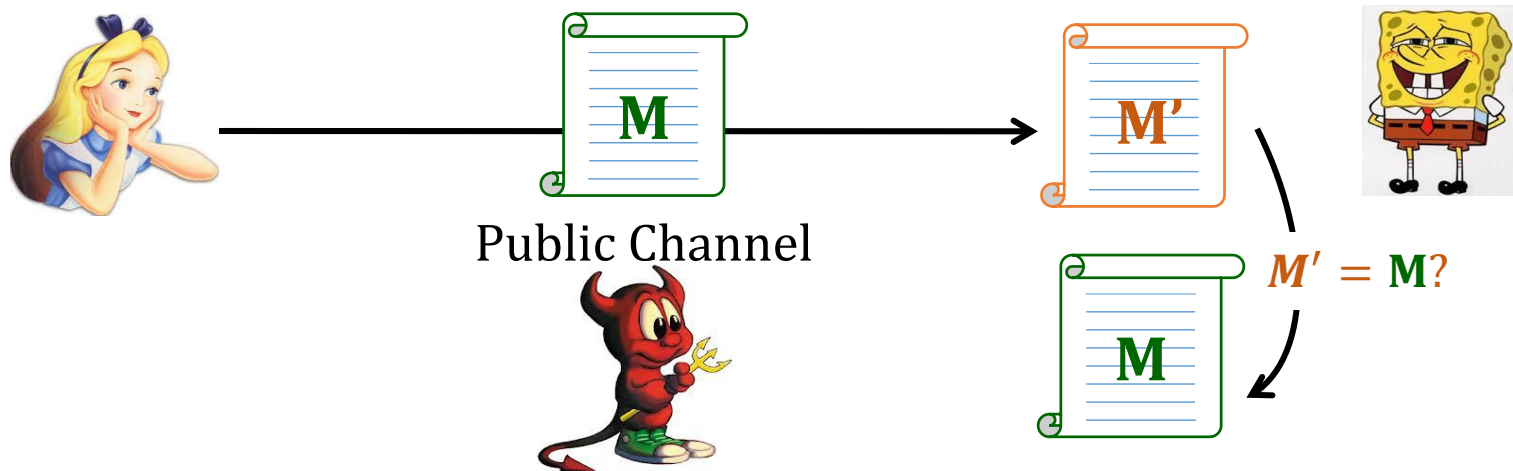
Crypto Hash Design& HMAC



# Problem:

- Alice wants to send Bob a message of  $n$  bits
- She wants Bob to verify that the message is the same one that was sent

Data Integrity



# Data Integrity (Motivation #1)

- **Integrity** — detect unauthorized writing (i.e., modification of data)
- Example: Inter-bank fund transfers
  - Confidentiality may be nice, **integrity is critical**
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc.

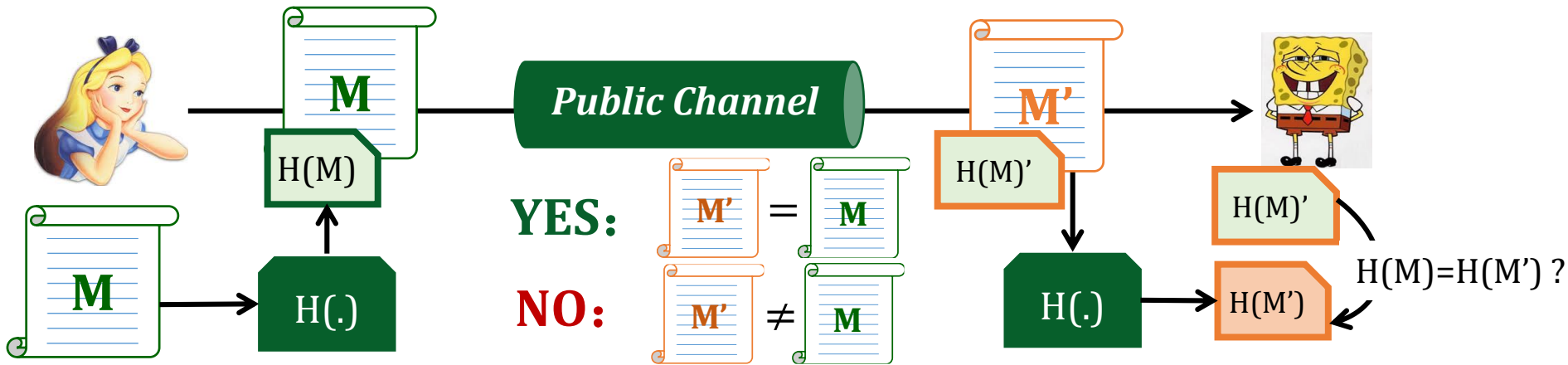
# Hash Function Motivation #2

- Suppose Alice signs  $M$ 
  - Alice sends  $M$  and  $S = [M]_{\text{Alice}}$  to Bob
  - Bob verifies that  $M = \{S\}_{\text{Alice}}$
  - Is it OK to just send  $S$ ?
- If  $M$  is big,  $[M]_{\text{Alice}}$  is costly to compute
- Suppose instead, Alice signs  $h(M)$ , where  $h(M)$  is much smaller than  $M$ 
  - Alice sends  $M$  and  $S = [h(M)]_{\text{Alice}}$  to Bob
  - Bob verifies that  $h(M) = \{S\}_{\text{Alice}}$

# Crypto Hash Function

- Crypto hash function  $h(x)$  must provide
  - **Compression** — output length is small
  - **Efficiency** —  $h(x)$  easy to compute for any  $x$
  - **One-way** — given a value  $y$  it is infeasible to find an  $x$  such that  $h(x) = y$  (*Pre-image resistant*)
  - **Weak collision resistance** — given  $x$  and  $h(x)$ , infeasible to find  $y \neq x$  such that  $h(y) = h(x)$ 
    - (*Second pre-image resistant*)
  - **Strong collision resistance** — infeasible to find any  $x$  and  $y$ , with  $x \neq y$  such that  $h(x) = h(y)$
  - Lots of collisions exist, but hard to find one

# Solution...



# Crypto Hash Design

- Desired property: **avalanche effect**
  - Change to 1 bit of input should affect about half of output bits
- Crypto hash functions consist of some number of rounds
- Want security and speed
  - Avalanche effect after few rounds
  - But simple rounds
- Analogous to design of block ciphers

# Popular Crypto Hashes

- **MD5** — invented by Rivest
  - 128 bit output
  - No longer secure. → replaced by SHA-256, AES-MAC
- **SHA-1** — A US government standard (similar to MD5)
  - 160 bit output
  - On February 23, 2017 CWI Amsterdam and Google announced they had performed a collision attack against SHA-1.
  - Replaced by SHA-2, SHA-3
- Many others hashes, but MD5 and SHA-1 most widely used

Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.

# Hash Uses

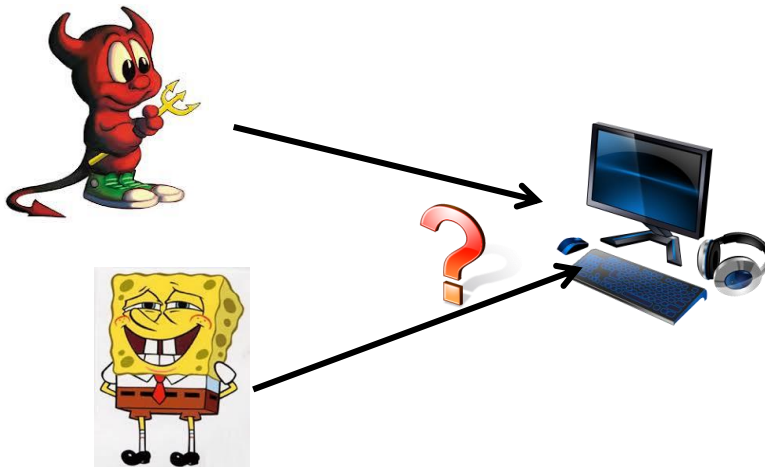
- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Data corruption detection
- Digital signature efficiency



# Password-based Authentication

# Authentication

- **Authentication** is the binding of an identity to a subject
- The external must provide information to enable the system to confirm its identity



# Password-based Authentication

- Passwords
- Lots of things act as passwords!
  - PIN
  - Social security number
  - Mother's maiden name
  - Date of birth
  - Name of your pet, etc.

# Good and Bad Passwords



## Bad Passwords

- frank
- Fido
- password
- 4444
- Pikachu
- 102560
- AustinStamp

## Good Passwords

- jflej,43j-EmmL+y
- 09864376537263
- P0kem0N
- FSa7Yago
- 0nceuP0nAt1m8
- PokeGCTall150

Good but hard  
to remember

Pokemon

Four score  
seven years ago

# Password Verification & Password File

- Bad idea to store passwords in a file
- But need a way to verify passwords
- Cryptographic solution: *hash* the passwords
  - Store  $y = h(\text{password})$
  - Can verify entered password by hashing
  - If Trudy obtains “password file,” she does not obtain passwords
- But Trudy can try a *forward search*
  - Guess  $x$  and check whether  $y = h(x)$
  - If so, Trudy has found password!



# Dictionary Attack

- Attacker pre-computes  $h(x)$  for all  $x$  in a dictionary of common passwords
- Suppose attacker get access to password file containing hashed passwords
  - Attacker only needs to compare hashes to his pre-computed dictionary
  - Same attack will work each time
- Can we prevent this attack? Or at least make attacker's job more difficult?

# Password File (Salted)



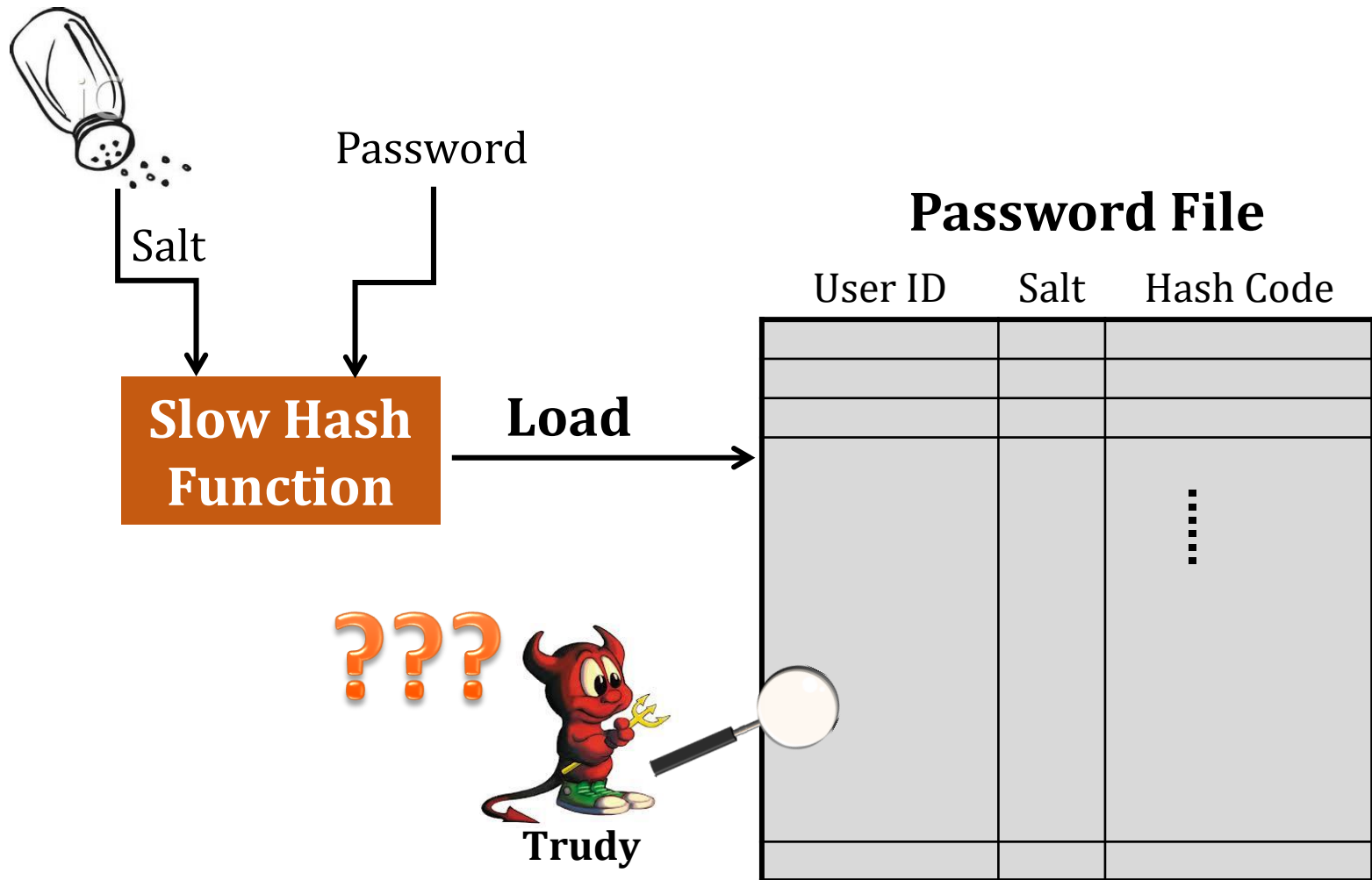
- Store hashed passwords
- Better to hash with *salt*
- Given password, choose random *s*, compute  $y = h(\text{password}, s)$ 
  - store the *pair*  $(s, y)$  in the password file
- Note: The salt *s* is not secret
- Easy to verify password
- Attacker must recompute dictionary hashes for each user → lots more work!

Randomize the input

Same pwd → diff. hashes

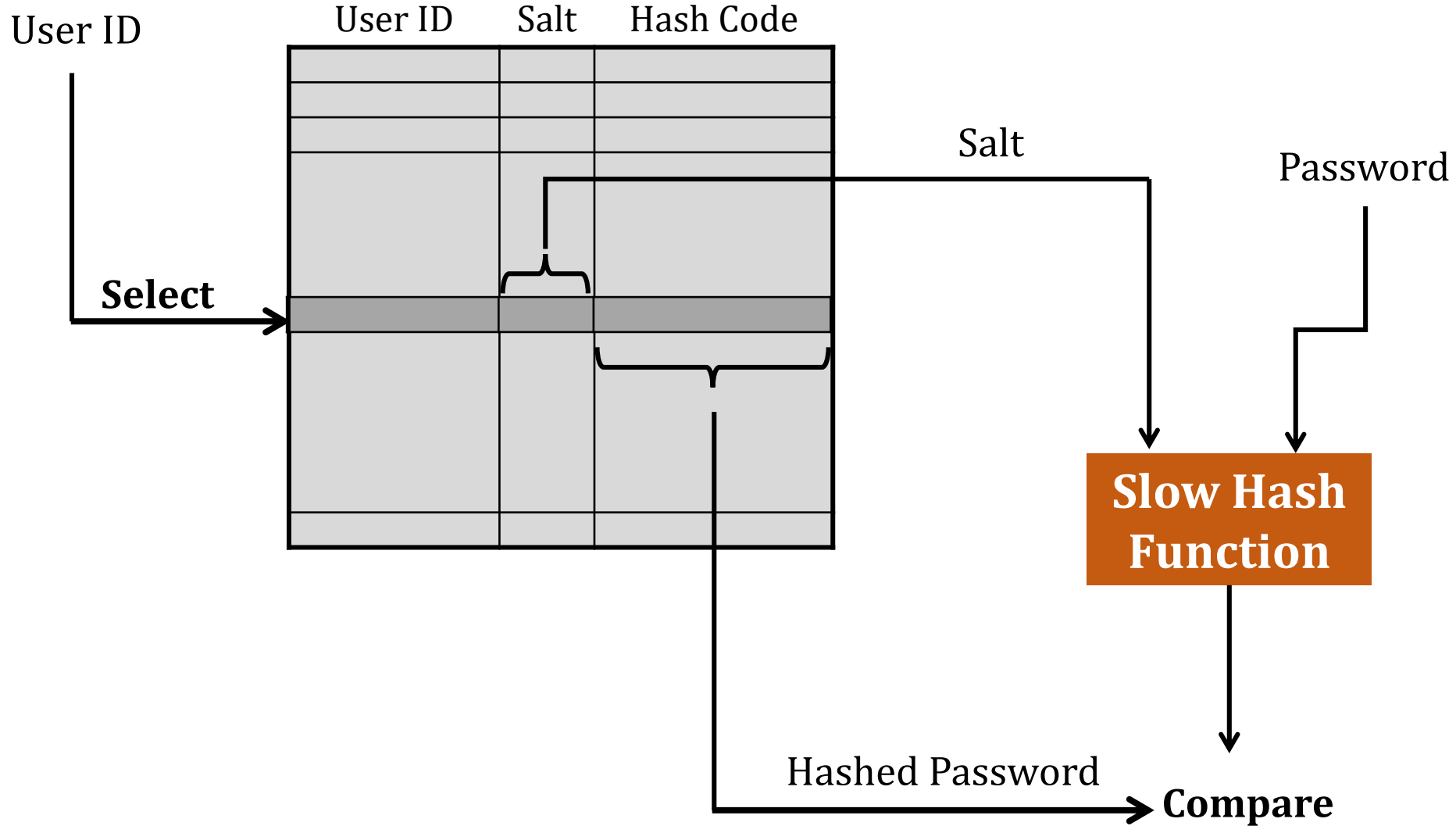
→ Un-expectable

# Loading a new password





# Password File



## Verifying a password

# Password Cracking Tools

- Popular password cracking tools
  - [Password Crackers](#)
  - [Password Portal](#)
  - [L0phtCrack and LC4](#) (Windows)
  - [John the Ripper](#) (Unix)
- *Admins should use these tools to test for weak passwords since attackers will*
- Good articles on password cracking
  - [Passwords - Cornerstone of Computer Security](#)
    - <http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password>
  - [Passwords revealed by sweet deal](#)
    - <http://news.bbc.co.uk/2/hi/technology/3639679.stm>

# Secure Real-World Protocol

-Secure Socket Layer (SSL) Protocol

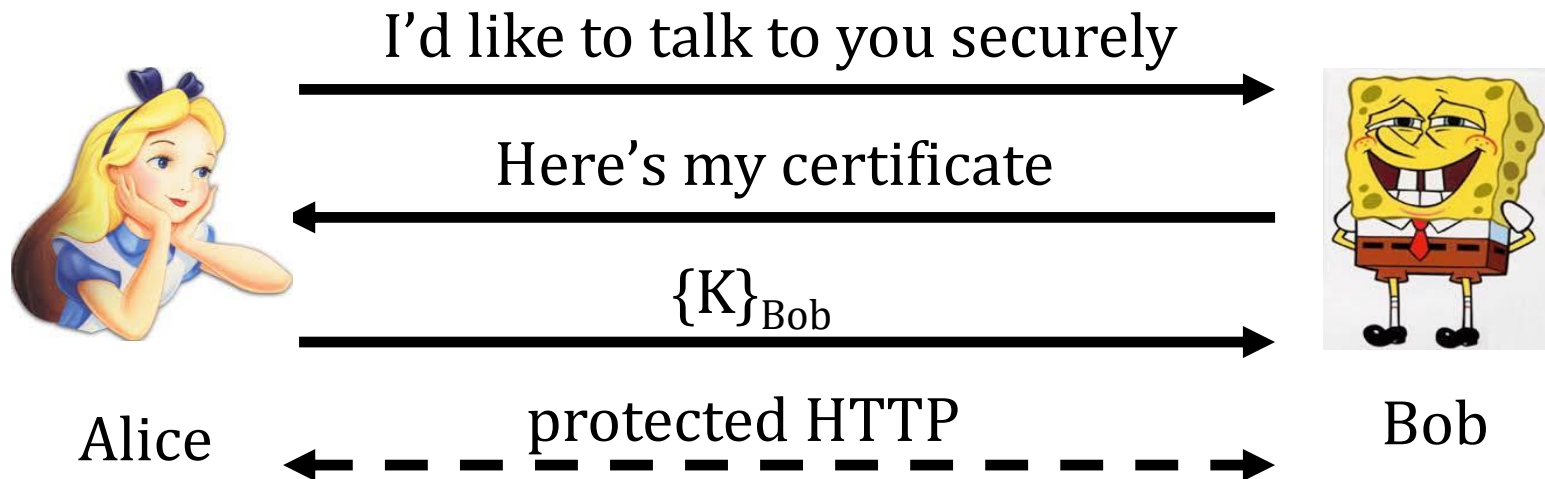
# What is SSL?

- SSL is the protocol used for majority of secure transactions on the Internet
- For example, if you want to buy a book at amazon.com...
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - As long as you have money, Amazon doesn't really care who you are
  - So, no need for mutual authentication



# Simple SSL-like Protocol

## General idea...



- Is Alice sure she's talking to Bob?

Bob is not explicitly authenticated and the only way Alice will know she is talking to Bob is when the encrypted data decrypts correctly. → which means Bob is the owner of the certificate as he announced before.

- Is Bob sure he's talking to Alice?

Alice is not authenticated to Bob at all, which is probably reasonable given the discussion above. (amazon example)

# SSL Authentication

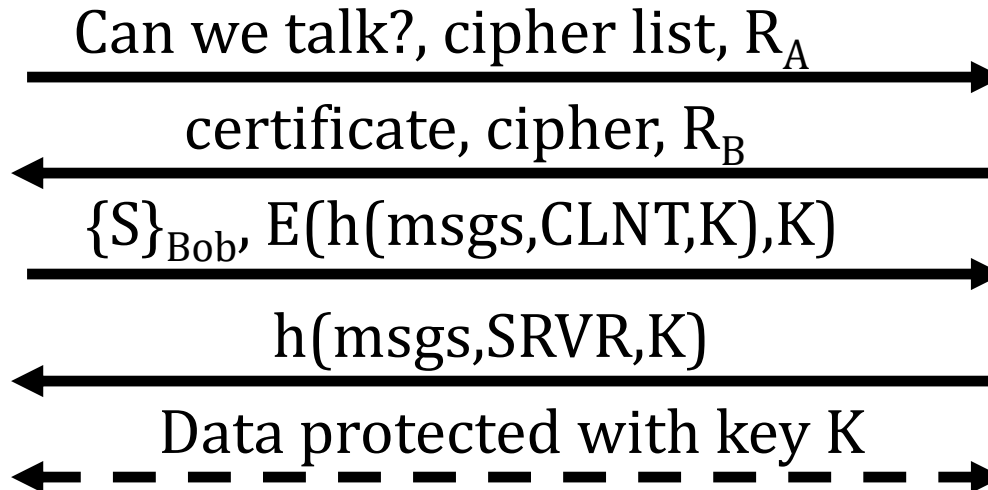
- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
  - Why would server not authenticate client?
- Mutual authentication is possible: Bob sends **certificate request** in message 2
  - Then client must have a valid certificate
  - If server wants to authenticate client, server could instead require password

# Simplified SSL Protocol

The actual SSL protocol is more complex



Alice

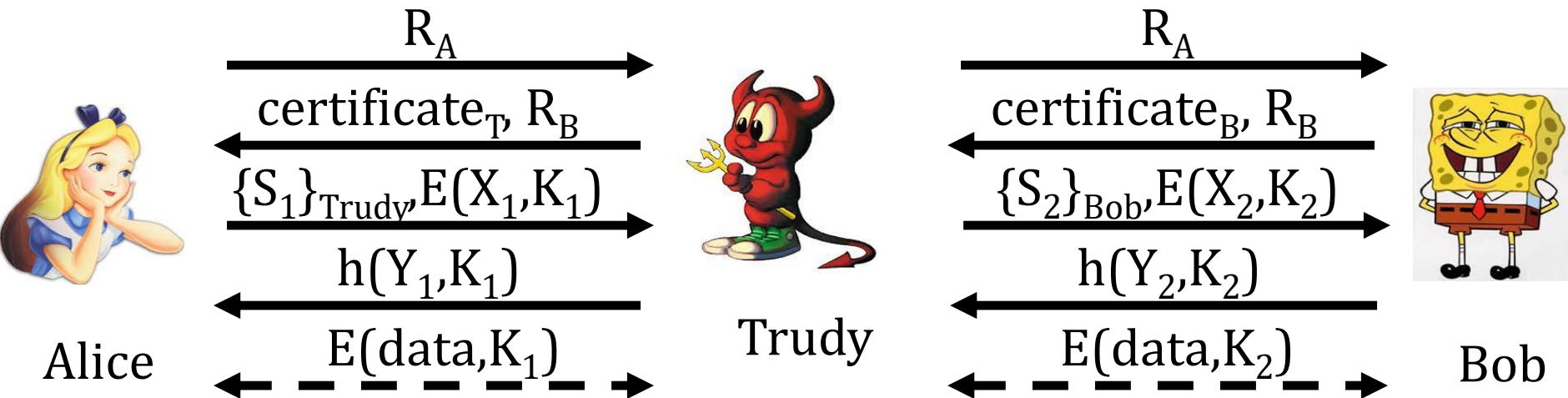


Bob

- S is known as **pre-master secret**
- $K = h(S, R_A, R_B)$
- “msgs” means all previous messages
- CLNT and SRVR are constants(literal string)

To verify that previous msgs have been received correctly

# SSL MiM Attack?



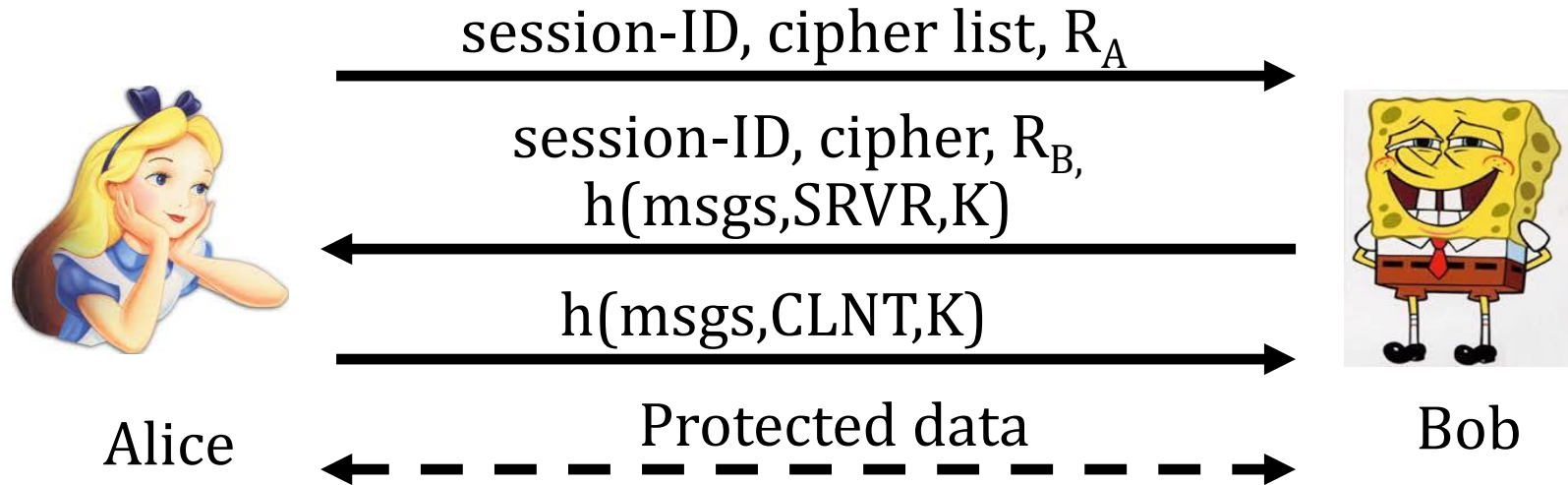
- **Q:** What prevents this MiM “attack”?
- **A:** Bob’s certificate must be signed by a certificate authority (CA)
- What does browser do if signature not valid?
- What does user do when browser complains?



# SSL Sessions vs Connections

- SSL **session** is established as shown on previous slides
- SSL designed for use with HTTP 1.0
- HTTP 1.0 often opens multiple simultaneous (parallel) **connections**
  - Multiple connections per session
- SSL session is costly, public key operations
- SSL has an efficient protocol for opening new connections ***given an existing session***

# SSL Connection



- Assuming SSL **session** exists
- So S is already known to Alice and Bob
- Both sides must remember session-ID
- Again,  $K = h(S, R_A, R_B)$

- Given one session, SSL can create any number of connections

❑ **No public key operations!** (relies on known S)

# Summary

- Cryptography is a awesome tool
  - But not a complete solution to security
  - Authenticity, Integrity, Secrecy
- **The #1 Rule**  
Never role your own crypto.  
(including inventing your own protocol)

# Computer Security



- How do we write software that can protect private information like  $K_e$ ,  $K_D$ ?
- How do we know implementation of ***E*** and ***D*** are correct?
- How do we build networks that are secure, reliable, and available?
- How do we ensure only Alice can access her keys?

The background of the slide is a full-page photograph of a vast, flat green field under a deep blue sky. The field is a lush green, and the sky is a clear, vibrant blue with some wispy white clouds. The horizon line is straight and divides the image roughly in half.

# Thank You!