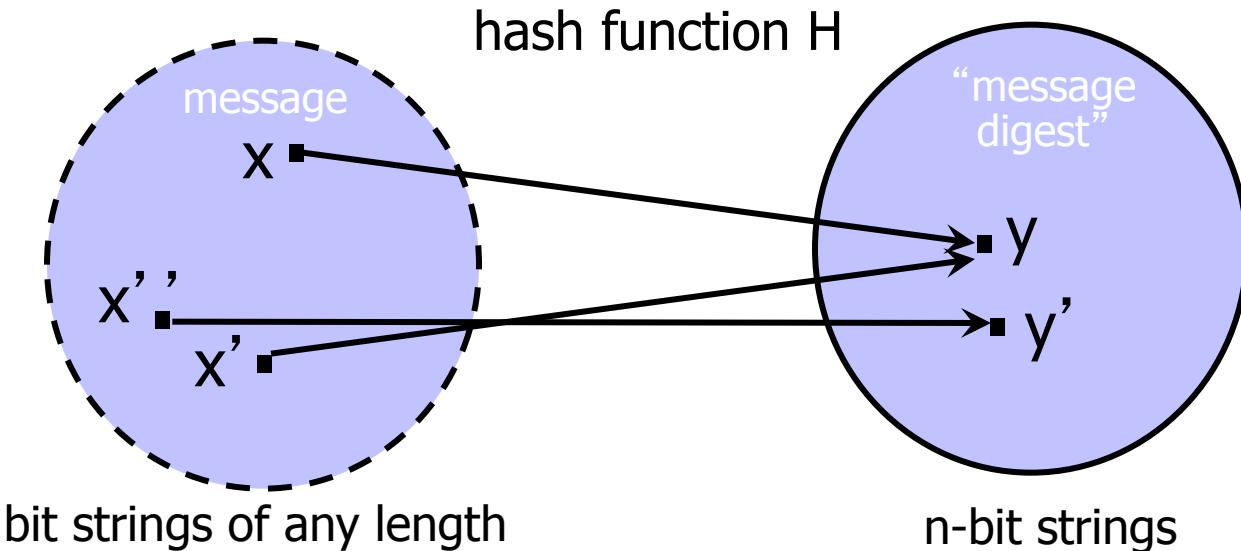


# Cryptographic Hash Functions

---

\*Slides borrowed from Vitaly Shmatikov

# Hash Functions: Main Idea



- ◆ Hash function  $H$  is a lossy compression function
  - Collision:  $H(x)=H(x')$  for some inputs  $x \neq x'$
- ◆  $H(x)$  should look “random”
  - Every bit (almost) equally likely to be 0 or 1
- ◆ A cryptographic hash function must have certain properties

# One-Way

---

## ◆ Intuition: hash should be hard to invert

- “Preimage resistance”
- Given a random  $y$ , it should be hard to find any  $x$  such that  $h(x)=y$ 
  - $y$  is an  $n$ -bit string randomly chosen from the output space of the hash function, i.e.,  $y=h(x')$  for some  $x'$

## ◆ How hard?

- Brute-force: try every possible  $x$ , see if  $h(x)=y$
- SHA-1 (a common hash function) has 160-bit output
  - Suppose we have hardware that can do  $2^{30}$  trials a pop
  - Assuming  $2^{34}$  trials per second, can do  $2^{89}$  trials per year
  - Will take  $2^{71}$  years to invert SHA-1 on a random image

# Birthday Paradox

---

- ◆  $T$  people
- ◆ Suppose each birthday is a random number taken from  $K$  days ( $K=365$ ) – how many possibilities?
  - $K^T$  - samples with replacement
- ◆ How many possibilities that are all different?
  - $(K)_T = K(K-1)\dots(K-T+1)$  - samples without replacement
- ◆ Probability of no repetition?
  - $(K)_T/K^T \approx 1 - T(T-1)/2K$
- ◆ Probability of repetition?
  - $O(T^2)$

# Collision Resistance

---

- ◆ Should be hard to find  $x \neq x'$  such that  $h(x) = h(x')$
- ◆ Birthday paradox
  - Let  $T$  be the number of values  $x, x', x'' \dots$  we need to look at before finding the first pair  $x \neq x'$  s.t.  $h(x) = h(x')$
  - Assuming  $h$  is random, what is the probability that we find a repetition after looking at  $T$  values?  $O(T^2)$
  - Total number of pairs?  $O(2^n)$ 
    - $n$  = number of bits in the output of hash function
  - Conclusion:  $T \approx O(2^{n/2})$
- ◆ Brute-force collision search is  $O(2^{n/2})$ , not  $O(2^n)$ 
  - For SHA-1, this means  $O(2^{80})$  vs.  $O(2^{160})$

# One-Way vs. Collision Resistance

---

## ◆ One-wayness does not imply collision resistance

- Suppose  $g()$  is one-way
- Define  $h(x)$  as  $g(x')$  where  $x'$  is  $x$  except the last bit
  - $h$  is one-way (cannot invert  $h$  without inverting  $g$ )
  - Collisions for  $h$  are easy to find: for any  $x$ ,  $h(x_0)=h(x_1)$

## ◆ Collision resistance does not imply one-wayness

- Suppose  $g()$  is collision-resistant
- Define  $h(x)$  to be  $0x$  if  $x$  is  $(n-1)$ -bit long, else  $1g(x)$ 
  - Collisions for  $h$  are hard to find: if  $y$  starts with 0, then there are no collisions; if  $y$  starts with 1, then must find collisions in  $g$
  - $h$  is not one way: half of all  $y$ 's (those whose first bit is 0) are easy to invert (how?), thus random  $y$  is invertible with prob.  $1/2$

# Weak Collision Resistance

---

- ◆ Given a randomly chosen  $x$ , hard to find  $x'$  such that  $h(x)=h(x')$ 
  - Attacker must find collision for a specific  $x$ ... by contrast, to break collision resistance, enough to find any collision
  - Brute-force attack requires  $O(2^n)$  time
- ◆ Weak collision resistance does not imply collision resistance (why?)

# Hashing vs. Encryption

---

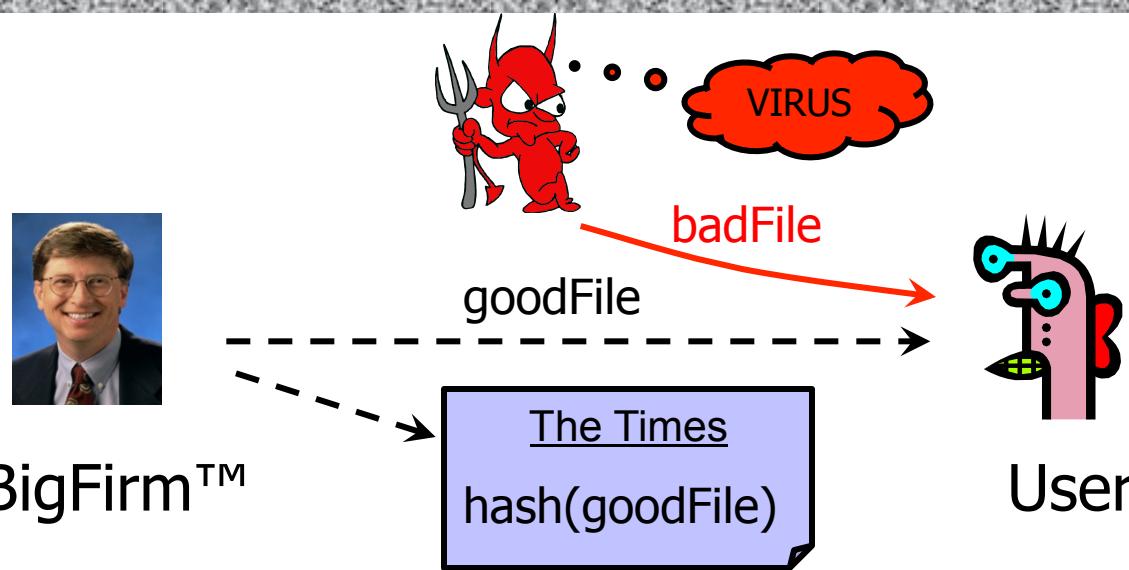
- ◆ Hashing is one-way. There is no “uh-hashing”!
  - A ciphertext can be decrypted with a decryption key... hashes have no equivalent of “decryption”
- ◆  $\text{Hash}(x)$  looks “random”, but can be compared for equality with  $\text{Hash}(x')$ 
  - Hash the same input twice → same hash value
  - Encrypt the same input twice → different ciphertexts
- ◆ Cryptographic hashes are also known as “cryptographic checksums” or “message digests”

# Application: Password Hashing

---

- ◆ Instead of user password, store **hash(password)**
- ◆ When user enters a password, compute its hash and compare with the entry in the password file
  - System does not store actual passwords!
  - Cannot go from hash to password!
- ◆ Why is hashing better than encryption here?
- ◆ Does hashing protect weak, easily guessable passwords?

# Application: Software Integrity



Software manufacturer wants to ensure that the executable file  
is received by users without modification...

Sends out the file to users and publishes its hash in the NY Times  
The goal is integrity, not secrecy

Idea: given  $\text{goodFile}$  and  $\text{hash}(\text{goodFile})$ ,  
very hard to find  $\text{badFile}$  such that  $\text{hash}(\text{goodFile}) = \text{hash}(\text{badFile})$

# Which Property Is Needed?

---

- ◆ Passwords stored as hash(password)
  - One-wayness: hard to recover entire password
  - Passwords are not random and thus guessable
- ◆ Integrity of software distribution
  - Weak collision resistance?
  - But software images are not random... maybe need full collision resistance
- ◆ Auctions: to bid B, send H(B), later reveal B
  - One-wayness... but does not protect B from guessing
  - Collision resistance: bidder should not be able to find two bids B and B' such that  $H(B)=H(B')$

# Common Hash Functions

---

- ◆ MD5
  - Completely broken by now

- ◆ RIPEMD-160
  - 160-bit variant of MD-5

- ◆ SHA-1 (Secure Hash Algorithm)

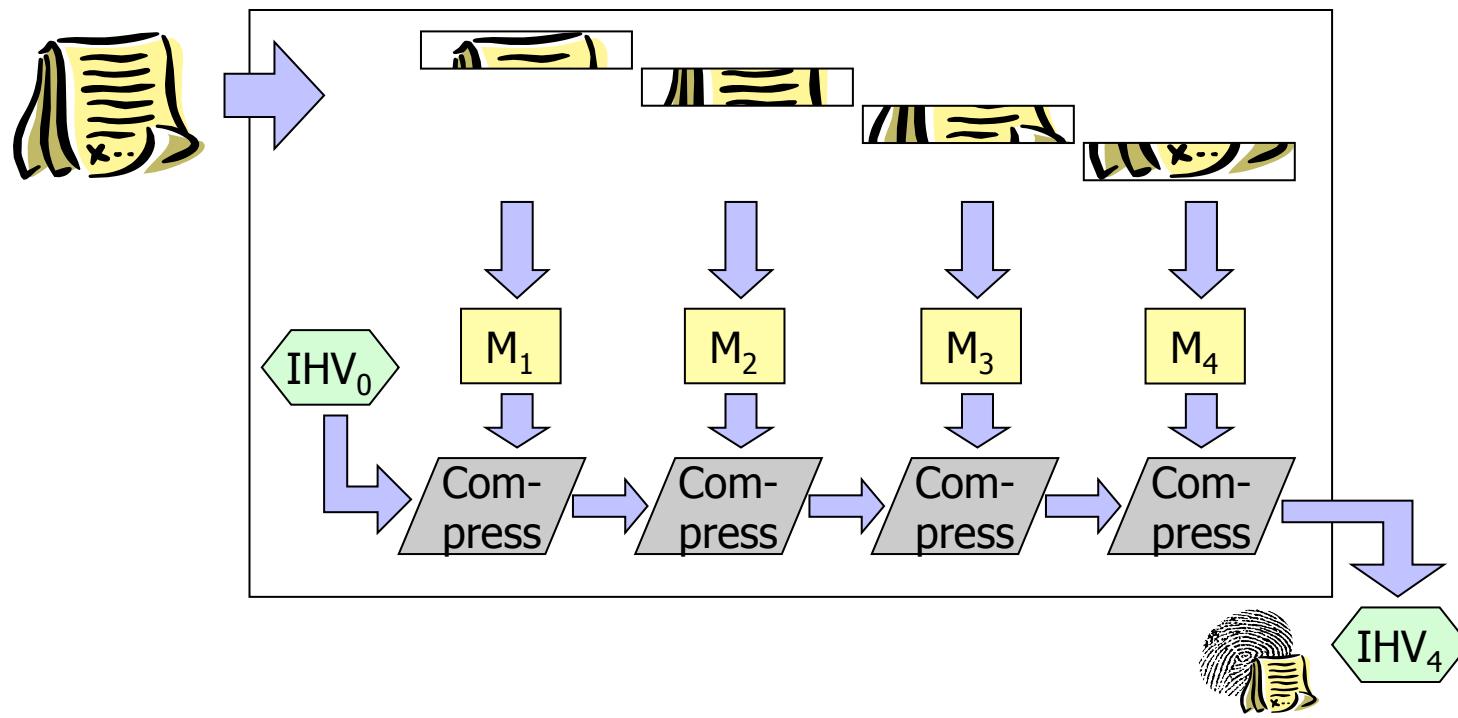
- Recently broken & deprecated

- 
  - ◆ SHA-256, SHA-3

- Still secure and recommended

# Overview of MD5

- ◆ Designed in 1991 by Ron Rivest
- ◆ Iterative design using compression function

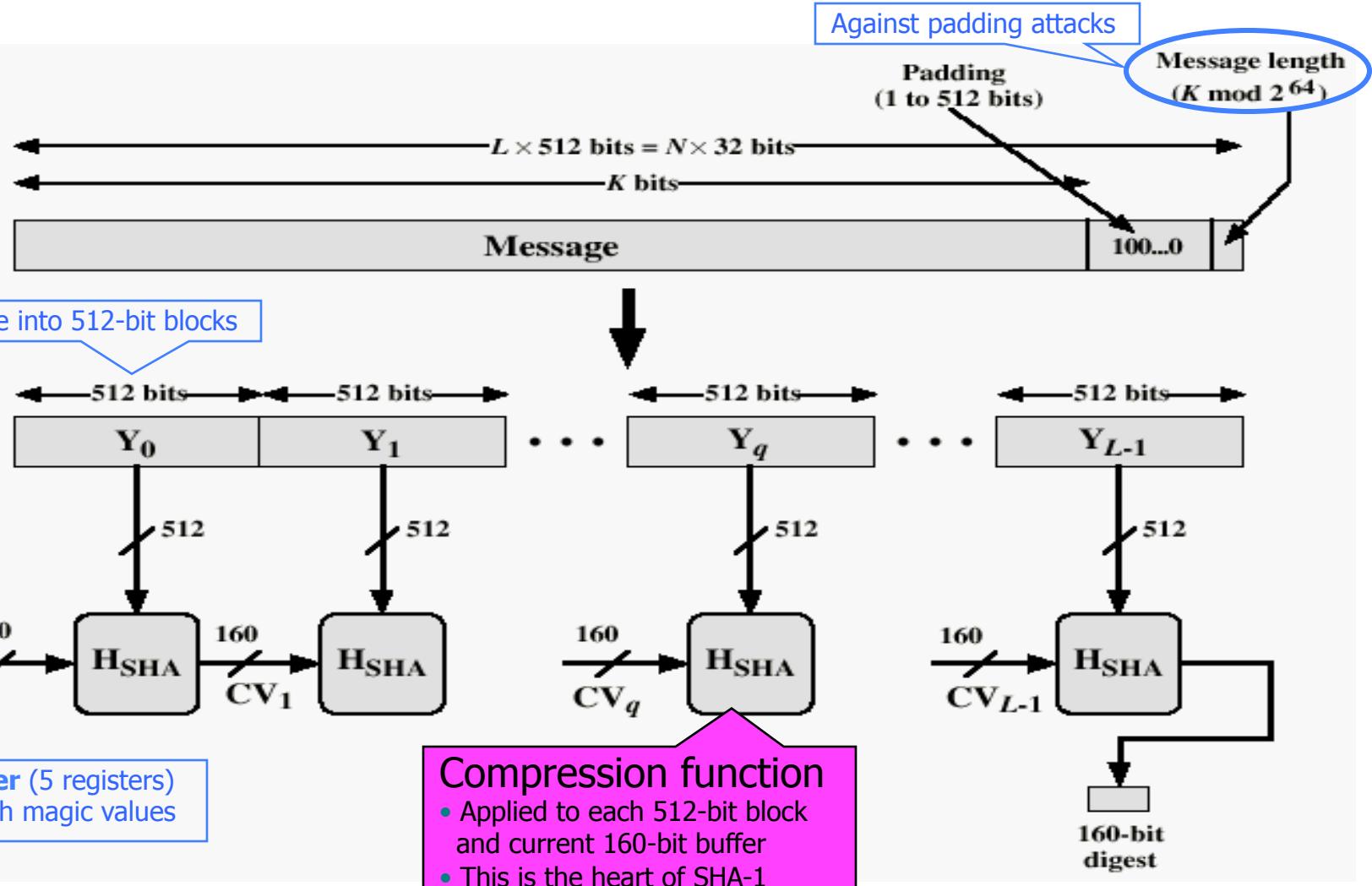


# History of MD5 Collisions

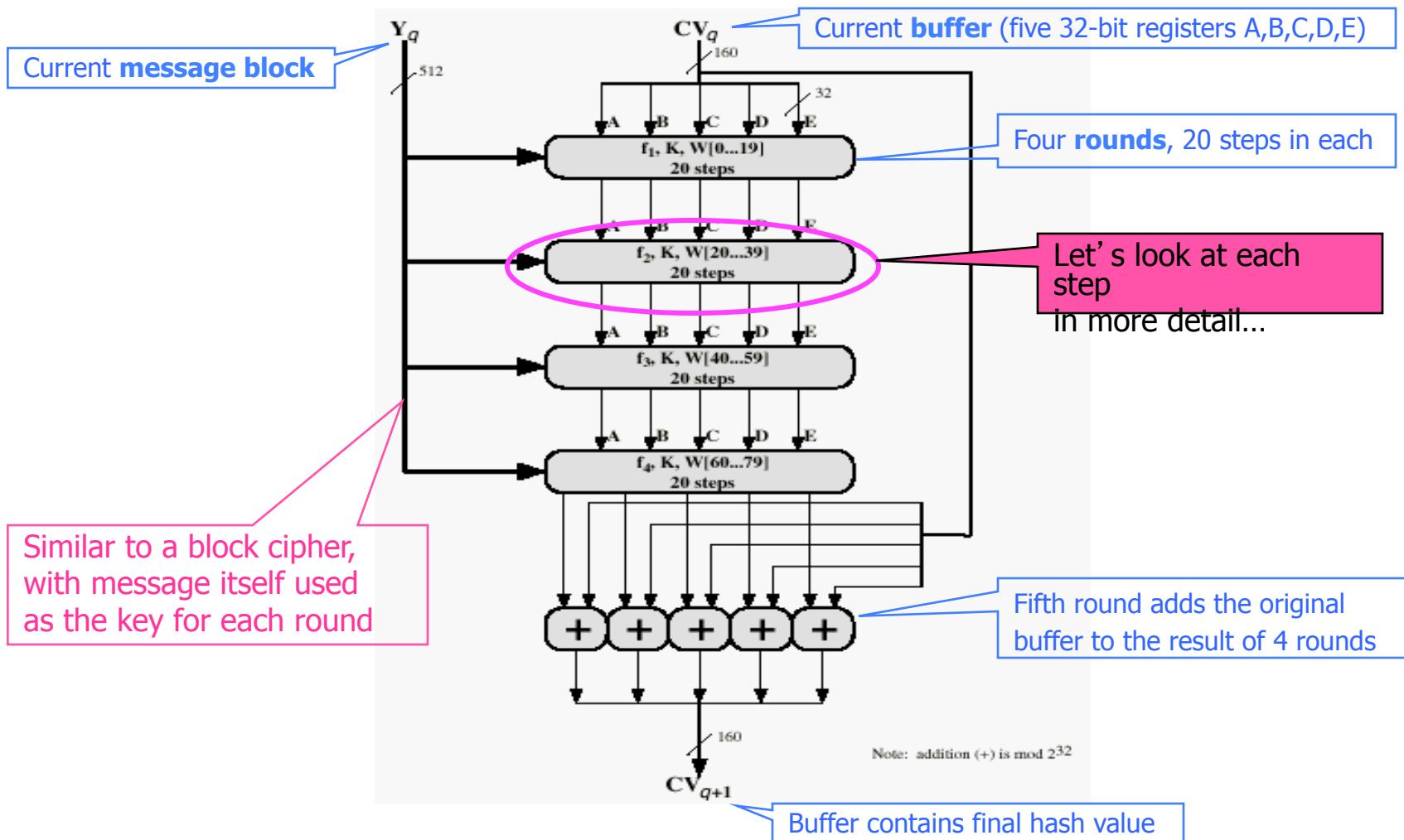
---

- ◆ 2004: first collision attack
  - The only difference between colliding messages is 128 random-looking bytes
- ◆ 2007: chosen-prefix collisions
  - For any prefix, can find colliding messages that have this prefix and differ up to 716 random-looking bytes
- ◆ 2008: rogue SSL certificates
  - Talk about this in more detail when discussing PKI
- ◆ 2012: MD5 collisions used in cyberwarfare
  - Flame malware uses an MD5 prefix collision to fake a Microsoft digital code signature

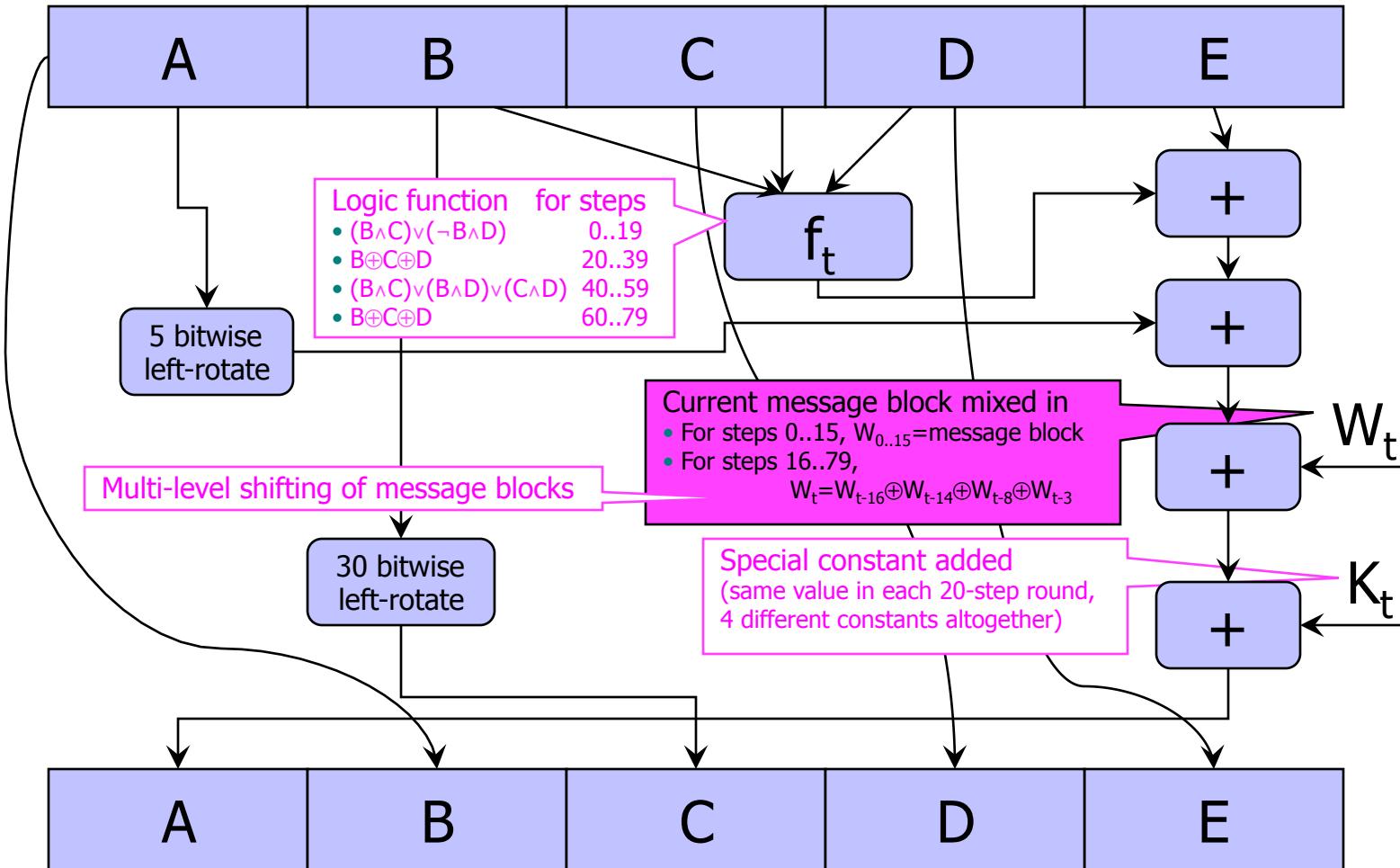
# Basic Structure of SHA-1



# SHA-1 Compression Function



# One Step of SHA-1 (80 steps total)



# How Strong Is SHA-1?

---

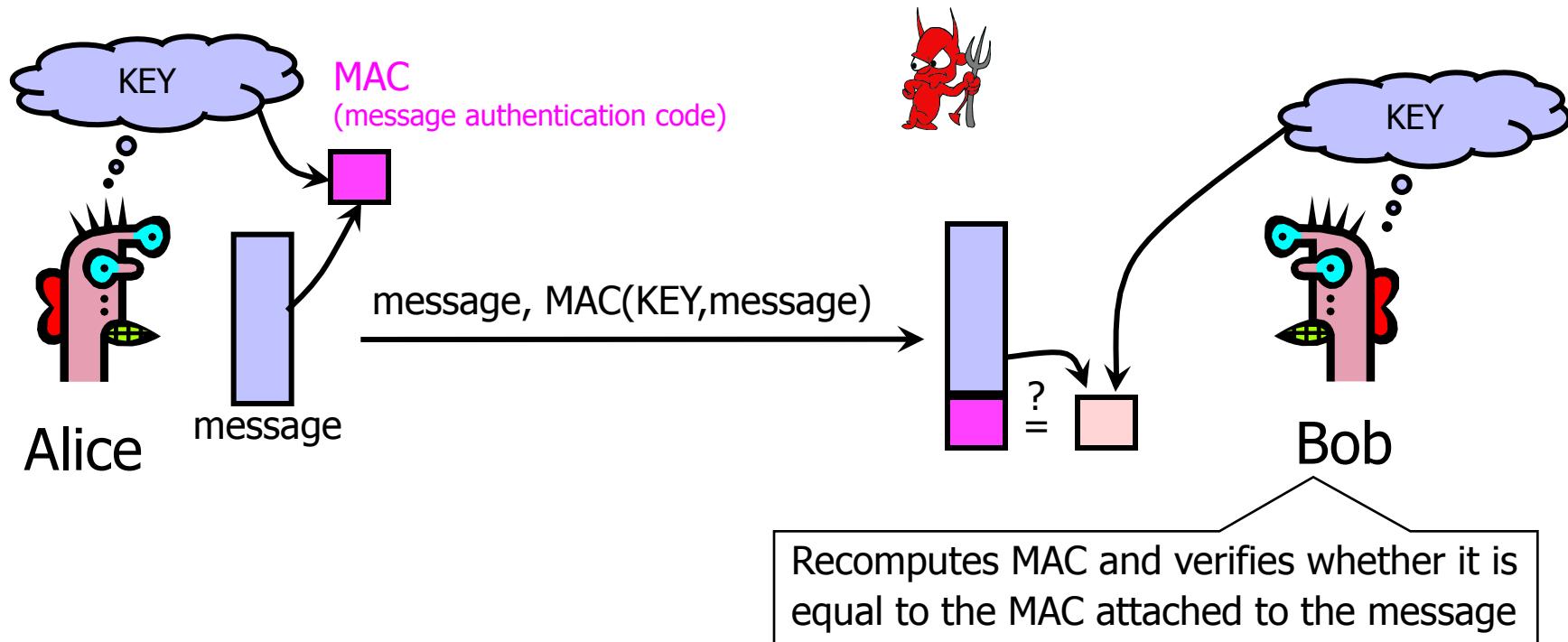
- ◆ Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- ◆ Brute-force inversion requires  $2^{160}$  ops, birthday attack on collision resistance requires  $2^{80}$  ops
- ◆ Weaknesses discovered in 2005
  - Collisions can be found in  $2^{63}$  ops
- ◆ Researchers at Google/CWI demonstrated first collision attack in 2017

# NIST Competition

---

- ◆ A public competition to develop a new cryptographic hash algorithm
  - Organized by NIST (read: NSA)
- ◆ 64 entries into the competition (Oct 2008)
- ◆ 5 finalists in 3<sup>rd</sup> round (Dec 2010)
- ◆ Winner: **Keccak** (Oct 2012)
  - standardized as SHA-3

# Integrity and Authentication



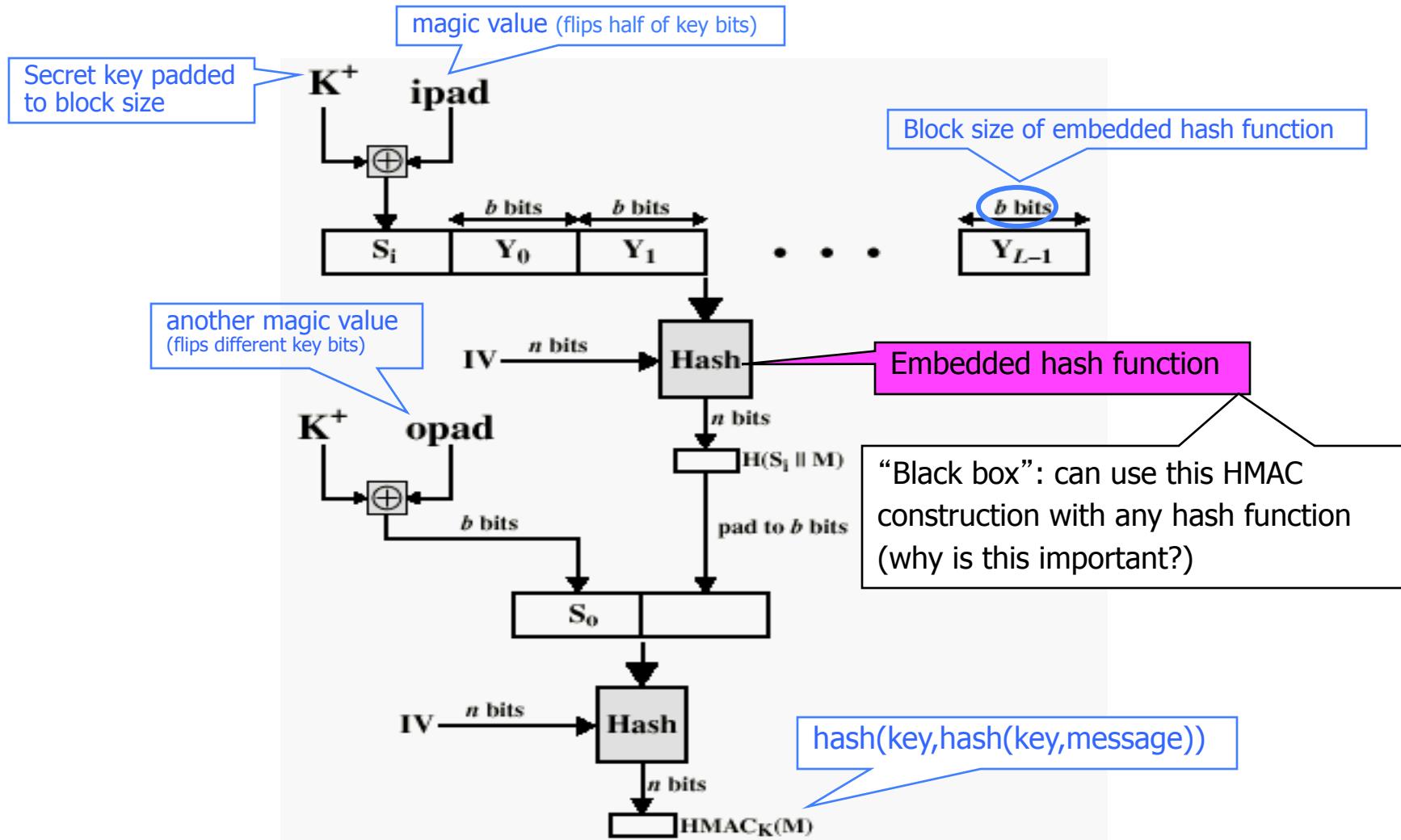
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message

# HMAC

---

- ◆ Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec
- ◆ Why not encryption?
  - Hashing is faster than encryption
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

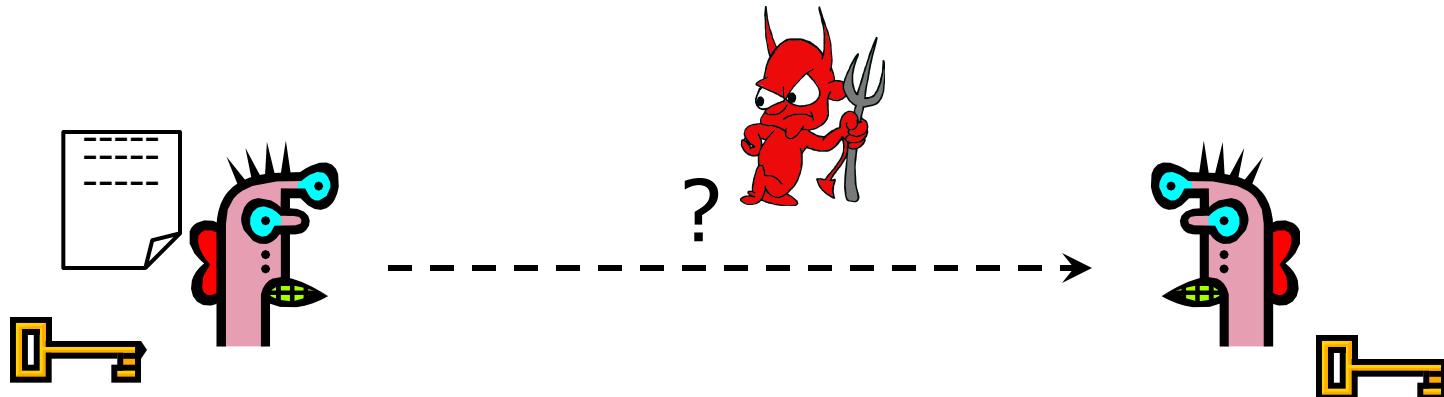
# Structure of HMAC



# Overview of Symmetric Encryption

---

# Basic Problem



Given: both parties already know the same **secret**

Goal: send a message confidentially

How is this achieved in practice?

Any communication system that aims to guarantee confidentiality must solve this problem

# Kerckhoffs's Principle

- ◆ An encryption scheme should be secure even if enemy knows everything about it except the key
  - Attacker knows all algorithms
  - Attacker does not know random numbers
- ◆ Do not rely on secrecy of the algorithms (“security by obscurity”)

Easy lesson:  
use a good random number generator!

Full name:  
Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof



# Randomness Matters!

The New York Times

Business Day  
Technology

WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY SCIENCE HEALTH SPORTS OPINION A

## Flaw Found in an Online Encryption Method

By JOHN MARKOFF

Published: February 14, 2012

SAN FRANCISCO — A team of European and American mathematicians and cryptographers have discovered an unexpected weakness in the encryption system widely used worldwide for online shopping, banking, e-mail and other Internet services intended to remain private and secure.

The flaw — which involves a small but measurable number of cases — has to do with the way the system generates random numbers, which are used to

 RECOMMEND

 TWITTER

 LINKEDIN

 COMMENTS  
(127)

 E-MAIL

 PRINT

 SINGLE PAGE

 REPRINTS

 MORE

### Readers' Comments

Readers shared their thoughts on this article.

[Read All Comments \(127\) »](#)

Internet users, there is nothing at sites will need to make changes to said

 Uptime

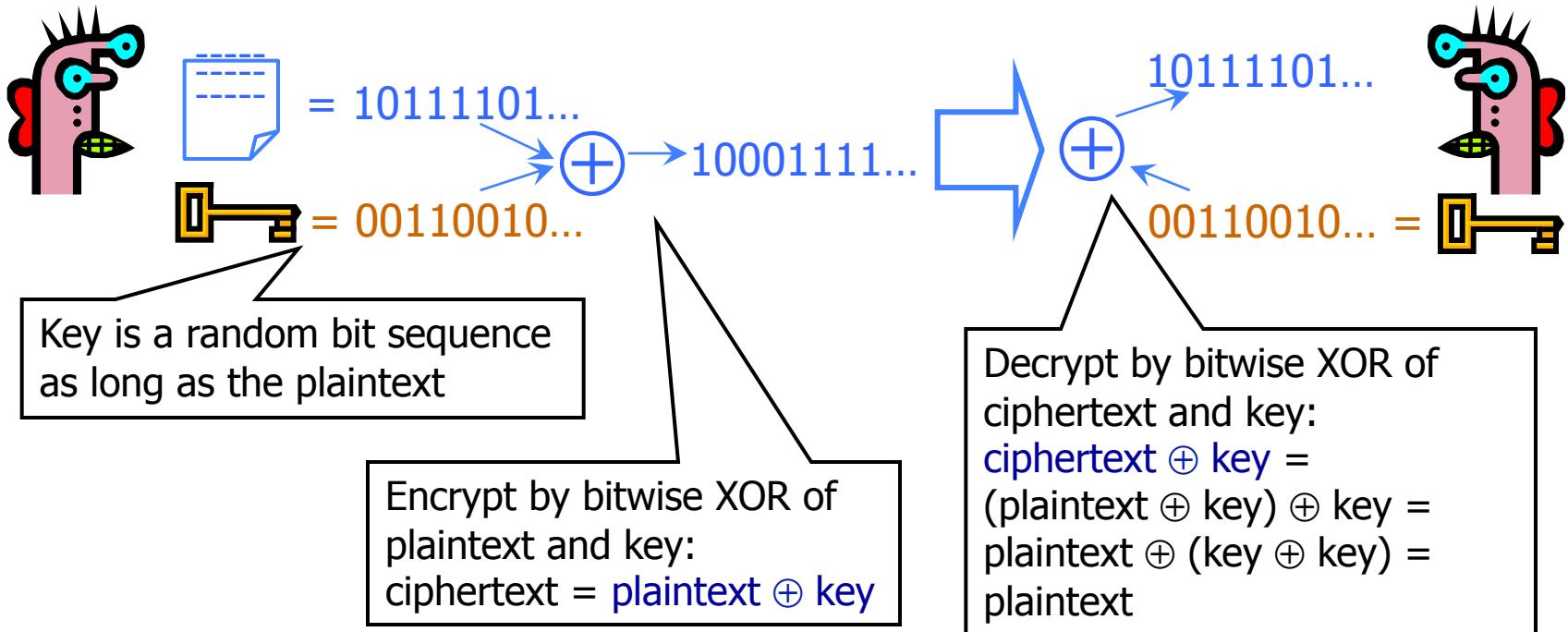
 IT and business computing insights



**Crypto shocker: four of every 1,000 public keys provide no security (updated)**

By Dan Goodin | Published 7 days ago

# One-Time Pad (Vernam Cipher)



Cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon, 1949)

# Advantages of One-Time Pad

---

- ◆ Easy to compute

- Encryption and decryption are the same operation
- Bitwise XOR is very cheap to compute

- ◆ As secure as theoretically possible

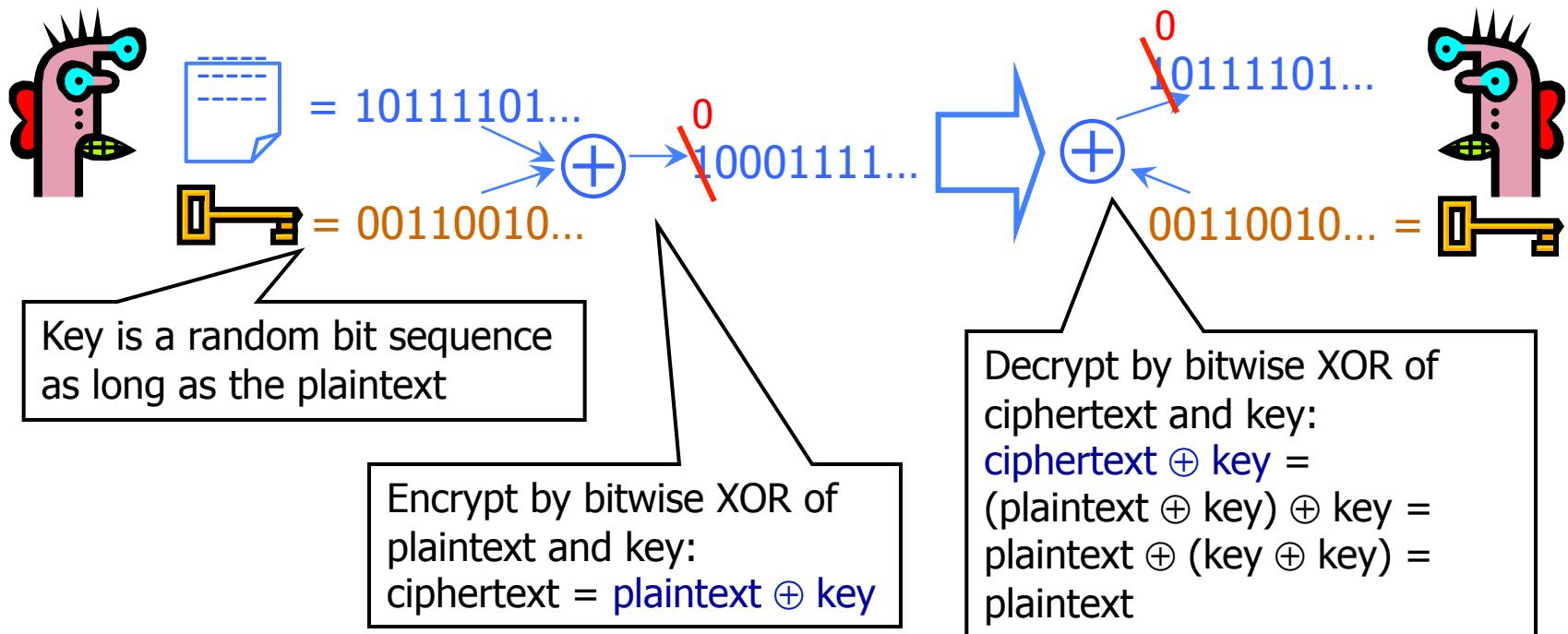
- Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
- ...if and only if the key sequence is truly random
  - True randomness is expensive to obtain in large quantities
- ...if and only if each key is as long as the plaintext
  - But how do the sender and the receiver communicate the key to each other? Where do they store the key?

# Problems with One-Time Pad

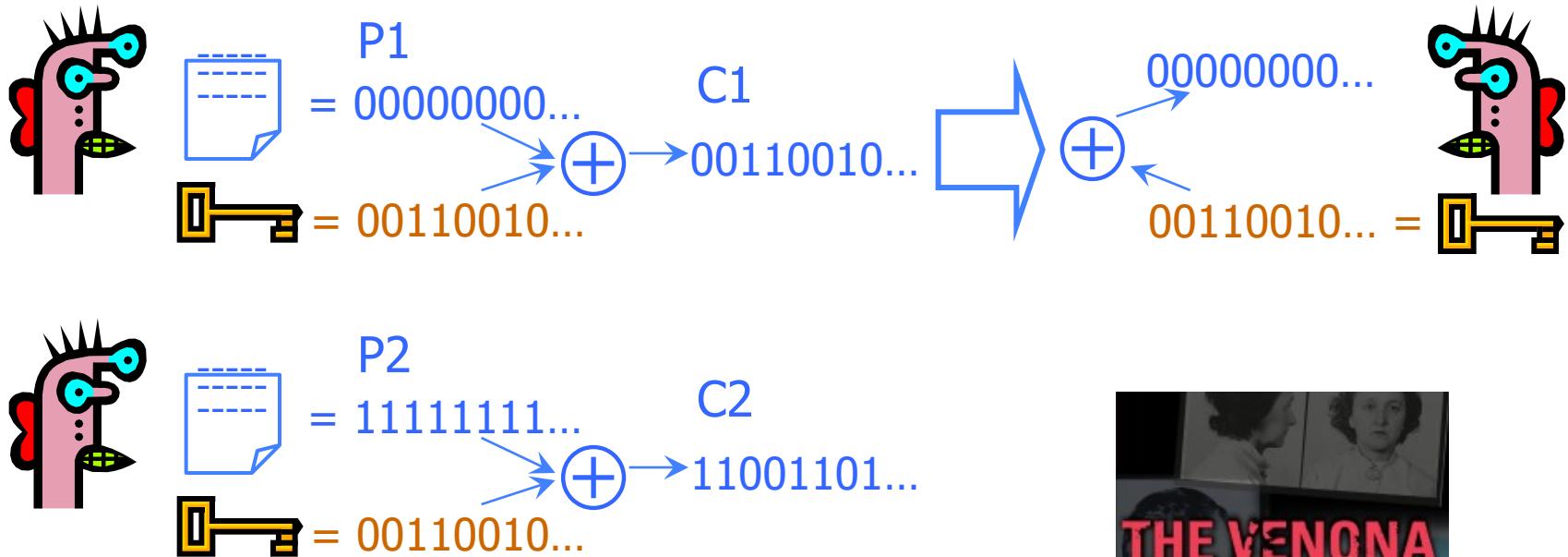
---

- ◆ Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- ◆ Does not guarantee integrity
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plaintext, but can easily change it to something else
- ◆ Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts

# No Integrity



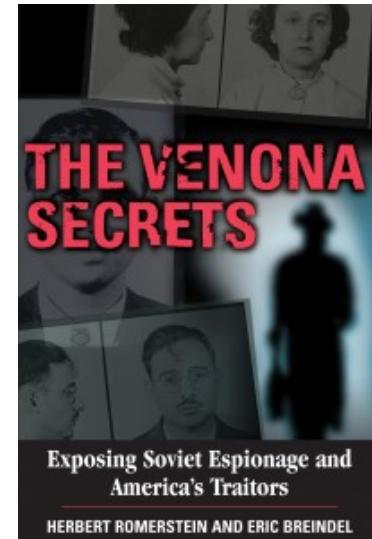
# Dangers of Reuse



Learn relationship between plaintexts

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) =$$

$$(P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2$$



# Reducing Key Size

---

- ◆ What to do when it is infeasible to pre-share huge random keys?
- ◆ Use special cryptographic primitives:  
**block ciphers, stream ciphers**
  - Single key can be re-used (with some restrictions)
  - Not as theoretically secure as one-time pad

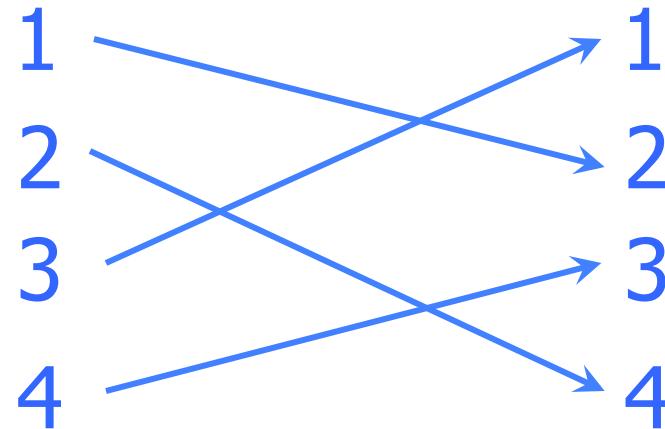
# Block Ciphers

---

- ◆ Operates on a single chunk (“block”) of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Same key is reused for each block (can use short keys)
- ◆ Result should look like a random permutation
- ◆ Not impossible to break, just very expensive
  - If there is no more efficient algorithm (unproven assumption!), can only break the cipher by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

# Permutation

---



CODE becomes DCEO

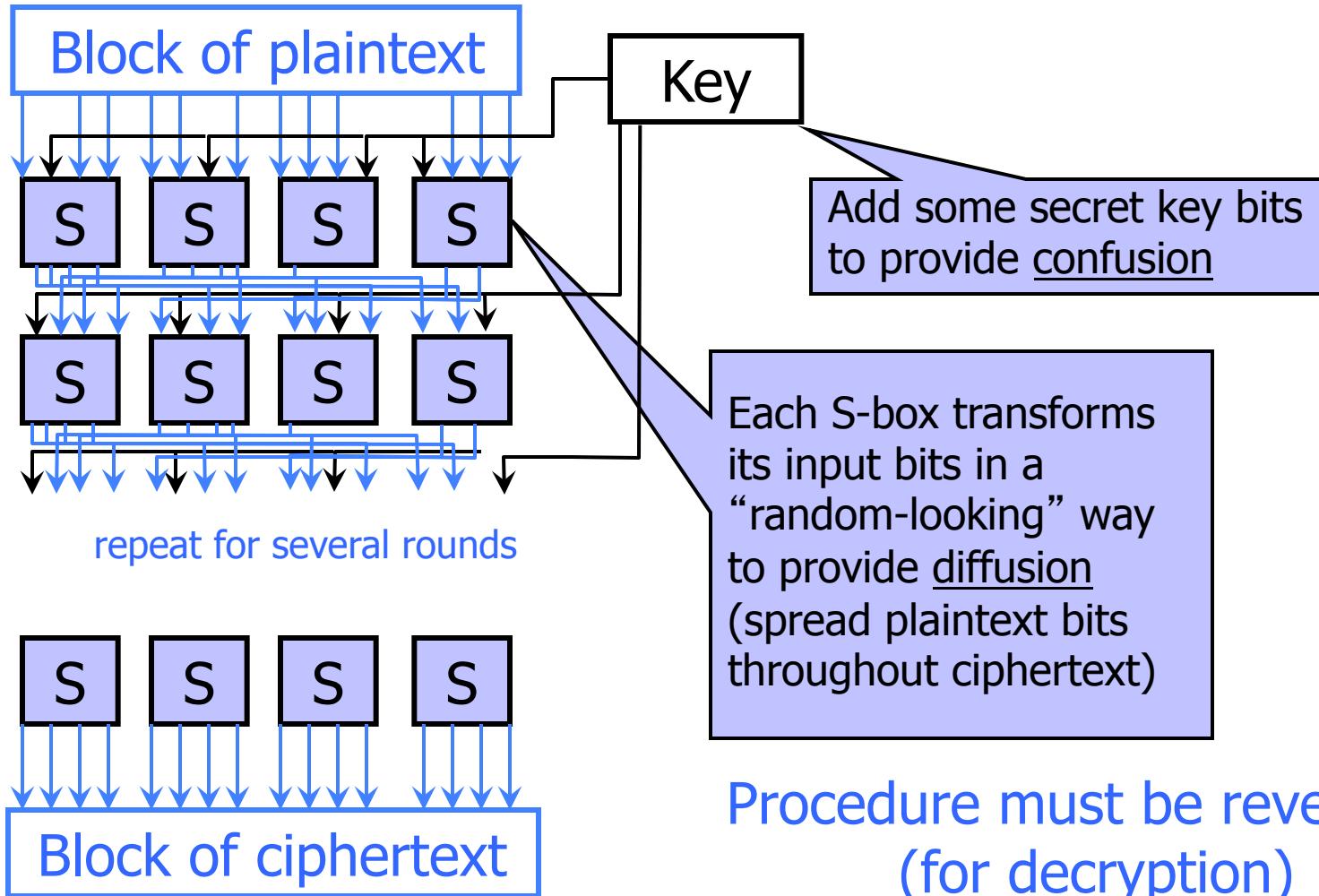
- ◆ For N-bit input,  $N!$  possible permutations
- ◆ Idea: split plaintext into blocks, for each block use secret key to pick a permutation, rinse and repeat
  - Without the key, permutation should “look random”

# A Bit of Block Cipher History

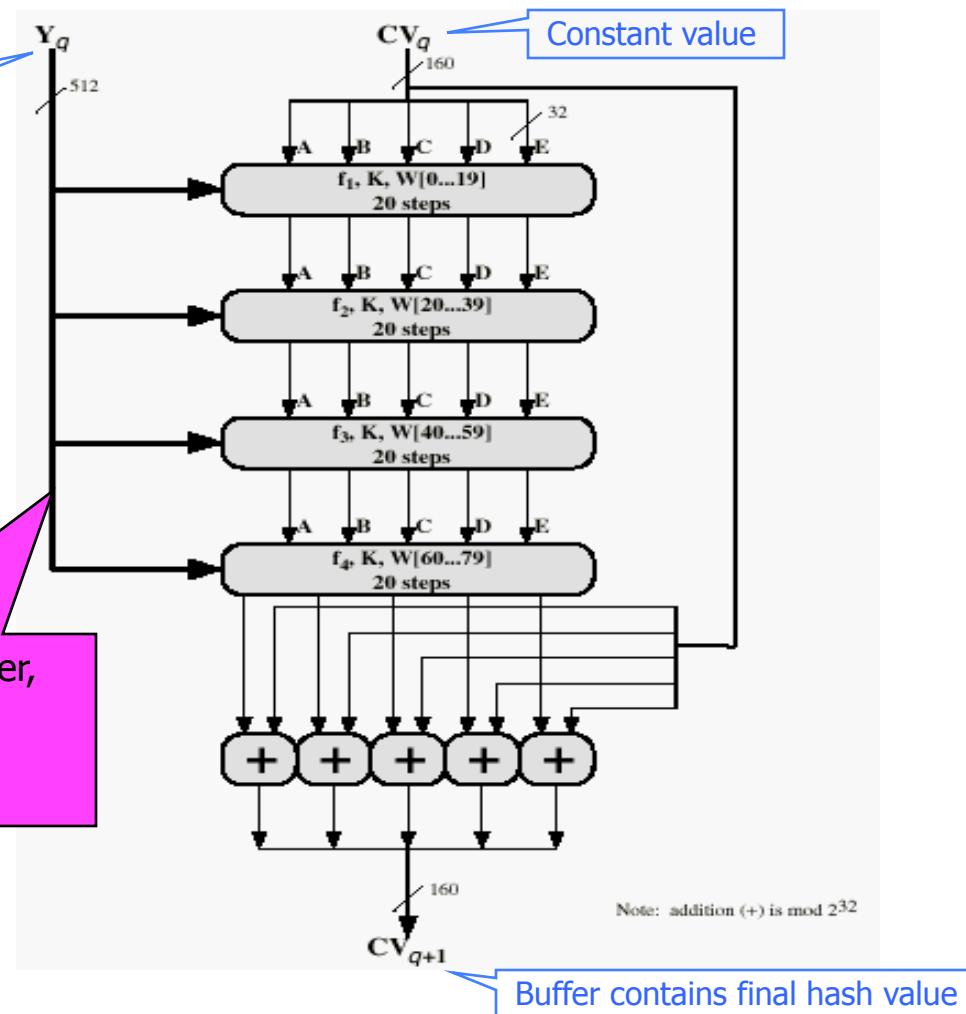
---

- ◆ Playfair and variants (from 1854 until WWII)
- ◆ Feistel structure Textbook
  - “Ladder” structure: split input in half, put one half through the round and XOR with the other half
  - After 3 random rounds, ciphertext indistinguishable from a random permutation
- ◆ DES: Data Encryption Standard Textbook
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity
  - Very widely used (usually as 3DES) until recently
    - 3DES: DES + inverse DES + DES (with 2 or 3 different keys)

# DES Operation (Simplified)



# Remember SHA-1?

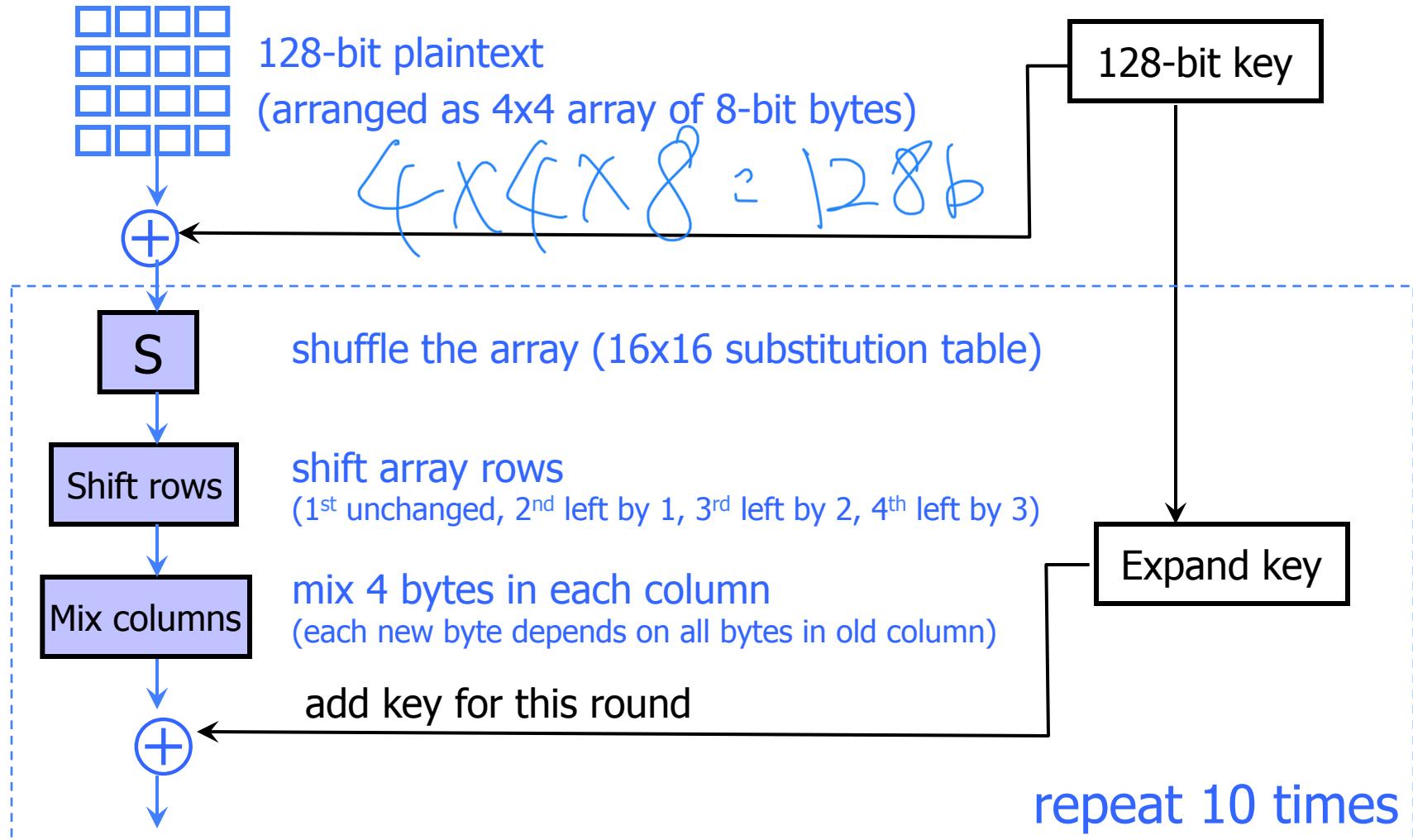


# Advanced Encryption Standard (AES)

---

- ◆ US federal standard as of 2001
- ◆ Based on the **Rijndael** algorithm
- ◆ 128-bit blocks, keys can be 128, 192 or 256 bits
- ◆ Unlike DES, does not use Feistel structure
  - The entire block is processed during each round
- ◆ Design uses some clever math
  - See section 8.5 of the textbook for a concise summary

# Basic Structure of Rijndael

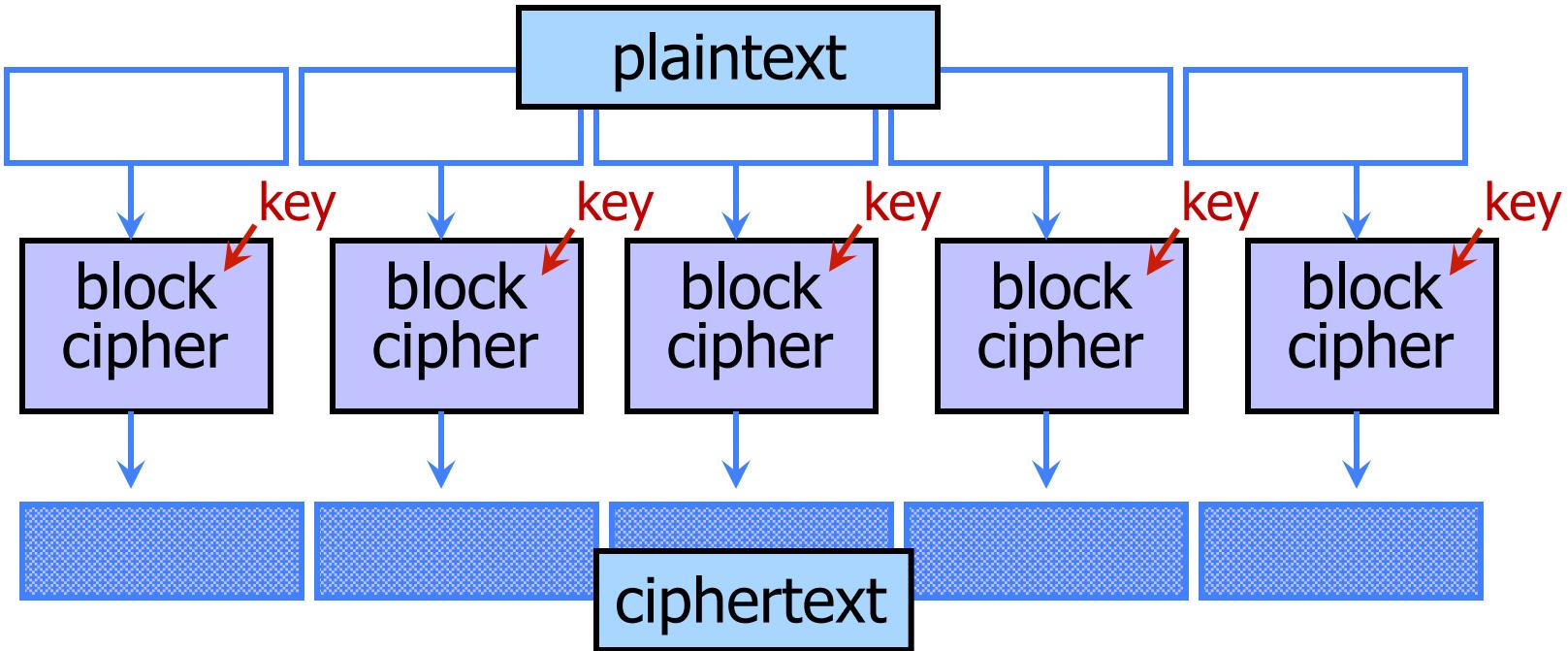


# Encrypting a Large Message

- ◆ So, we've got a good block cipher, but our plaintext is larger than 128-bit block size
  - ◆ Electronic Code Book (ECB) mode
    - Split plaintext into blocks, encrypt each one separately using the block cipher
  - ◆ Cipher Block Chaining (CBC) mode
    - Split plaintext into blocks, XOR each block with the result of encrypting previous blocks
  - ◆ Also various counter modes, feedback modes, etc.
- leak info*
- Good*

# ECB Mode

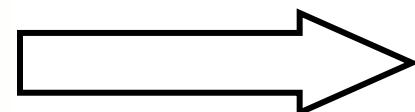
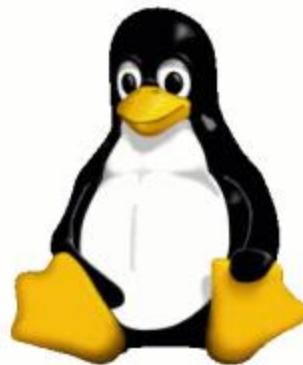
Electronic Code Block



- ◆ Identical blocks of plaintext produce identical blocks of ciphertext
- ◆ No integrity checks: can mix and match blocks

# Information Leakage in ECB Mode

[Wikipedia]



Encrypt in ECB mode



# Adobe Passwords Stolen (2013)

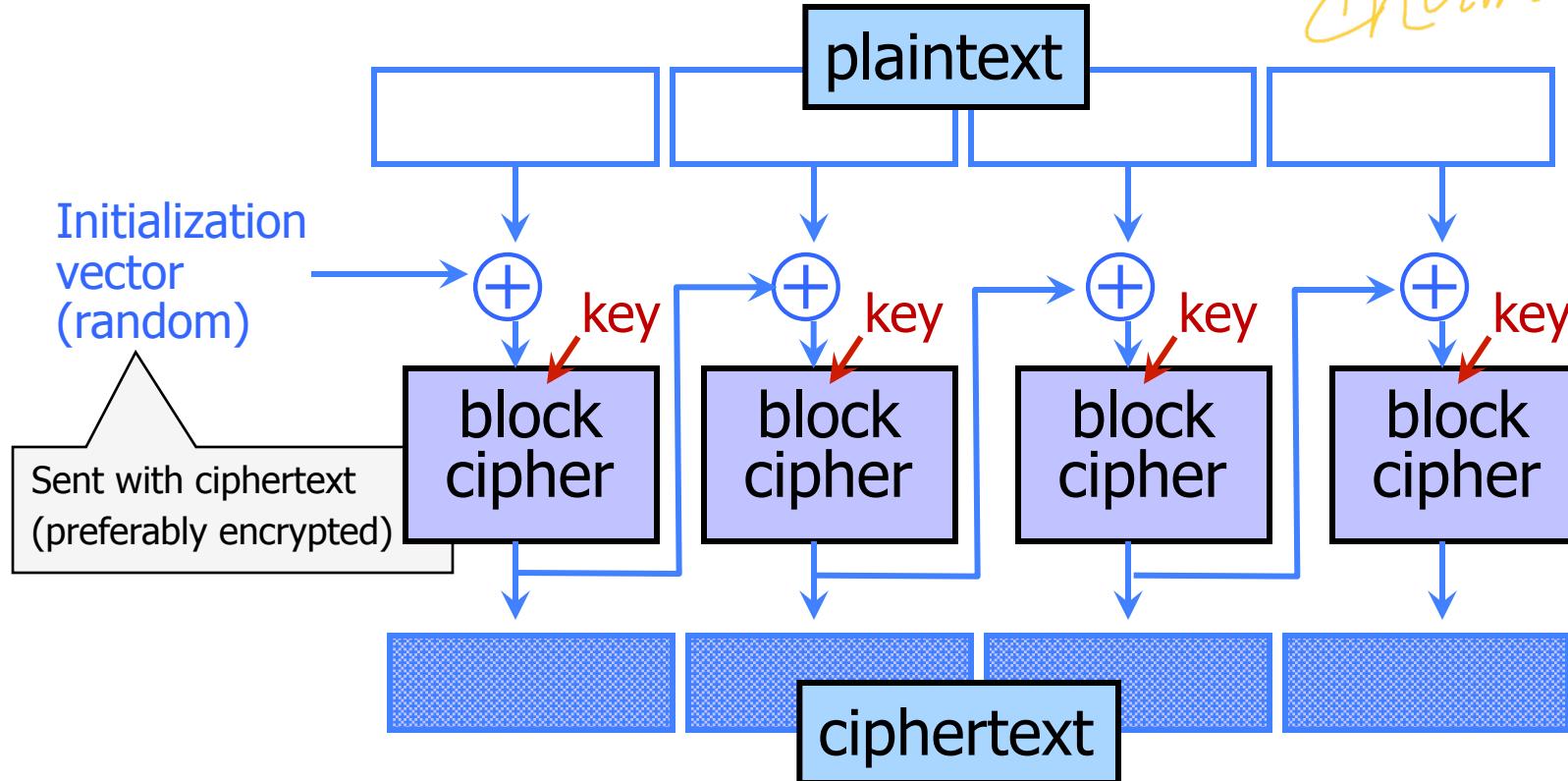
- ◆ 153 million account passwords
  - 56 million of them unique
- ◆ Encrypted using 3DES in ECB mode rather than hashed

```
+10010410 | |          a@ic.fbi.gov | -007j71B002E51Q3p47QBW=-| -city|  
79985232-| --|- a@fbi.gov-| -+ujcil90fBnioxG6CatHBw=-| -anniversary| --  
105009730-| --|- gon@ic.fbi.gov-| -9nCgb38RHiw=-| -band| --  
108684532-| --|- burn@ic.fbi.gov-| -EQ7fIpT7i/Q=-| -numbers| --  
63041670-| --|- iv-|-hRwtmq98mKzioxG6CatHBw=-| -|-  
94038395-| --|- n@ic.fbi.gov-| -MreVpEovYi7ioxG6CatHBw=-| -eod date| --  
116097938-| --|- -|-Tur7Wt2zH5CwIIHfjvcHKQ=-| -SH?|-  
83310434-| --|- c.fbi.gov-| -NLupdfyYrsM=-| -ATP MIDDLE| --  
113389790-| --|- iv-|-iMhaearHXjPioxG6CatHBw=-| -w|-  
113931981-| --|- @ic.fbi.gov-| -ltmosXxYnP3ioxG6CatHBw=-| -See MSDN| --  
114081741-| --|- lom@ic.fbi.gov-| -ZcDbLlvCad0=-| -fuzzy boy 20| --  
106145242-| --|- @ic.fbi.gov-| -xc2KumNGzYfioxG6CatHBw=-| -4s| --  
106437837-| --|- i.gov-| -adIewKvmJEsFqx0HFoFrwg=-| -|-  
96649467-| --|- ius@ic.fbi.gov-| -lsYw5KRkNT/ioxG6CatHBw=-| -glass o  
96670195-| --|- .fbi.gov-| -X4+k4uhyhD/ioxG6CatHBw=-| -|-  
105095956-| --|- earthlink.net-| -ZU2tTTFIZq/ioxG6CatHBw=-| -socialsecurity#| --  
108260815-| --|- r@genext.net-| -MuKnZ7KtsiHioxG6CatHBw=-| -socialsecurity| --  
83508352-| --|-h ghotmail.com-| -ADEcoaN2oUM=-| -socialsecurityno.| --  
83023162-| --|-k 590@aol.com-| -9HT+kVHQfs4=-| -socialsecurity name| --  
90331688-| --|-b .edu-| -nNiWEcoZTBmXrIXpAZiRHQ=-| -ssn#| --  
]  
~
```

>Password hints

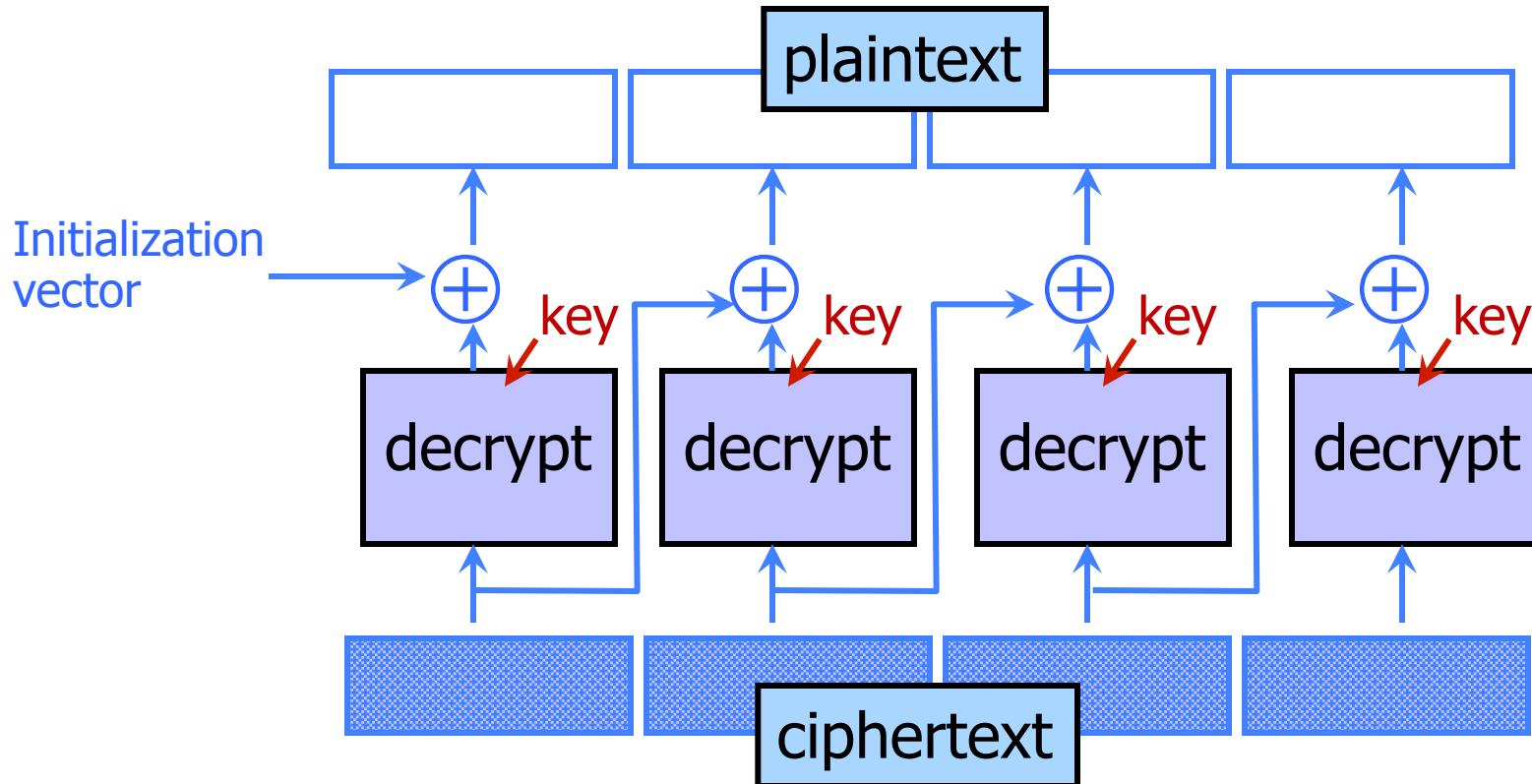
# CBC Mode: Encryption

Cipher block  
chaining-



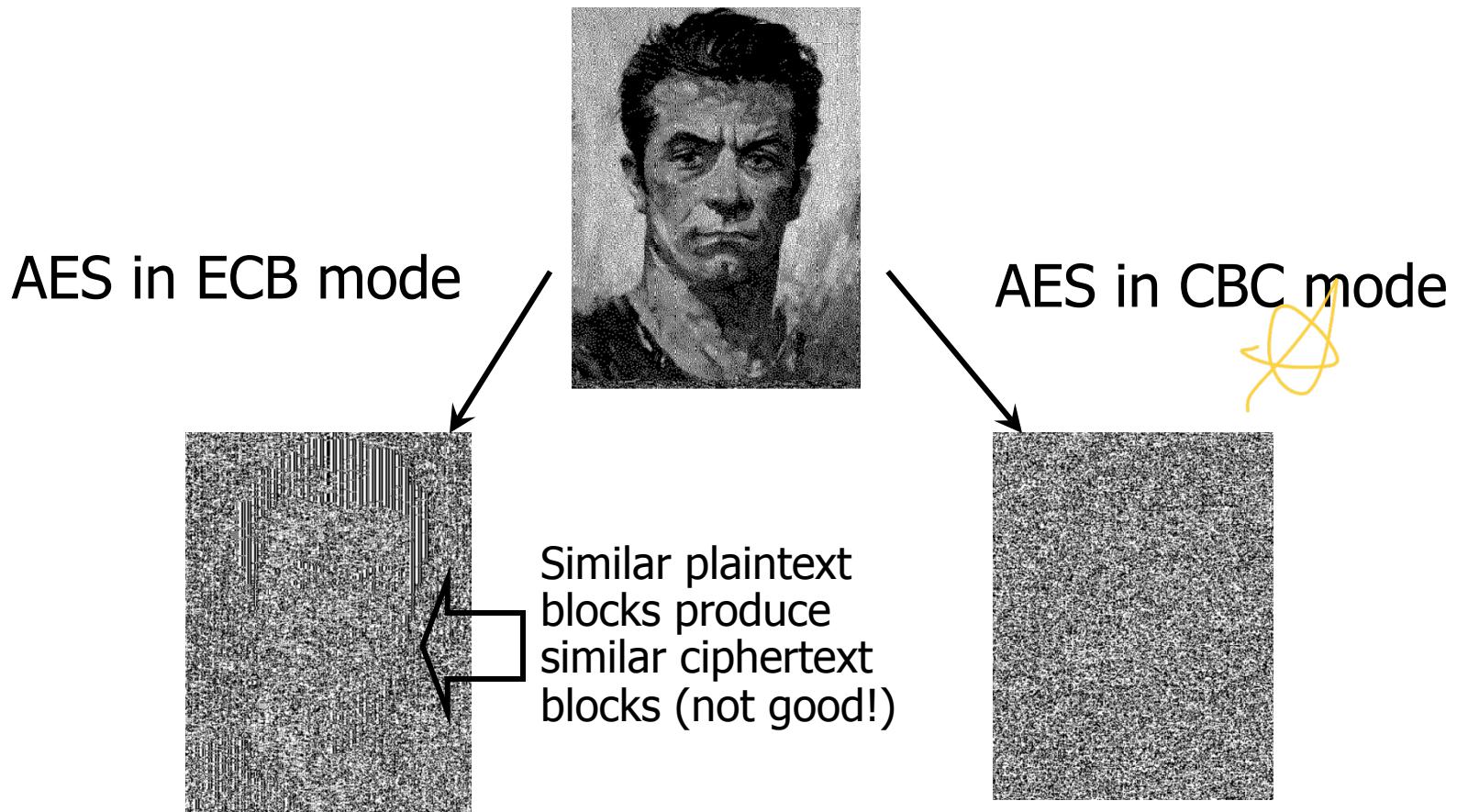
- ◆ Identical blocks of plaintext encrypted differently
- ◆ Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

# CBC Mode: Decryption



# ECB vs. CBC

[Picture due to Bart Preneel]



# Choosing the Initialization Vector

---

- ◆ Key used only once

- No IV needed (can use IV=0)

- ◆ Key used multiple times

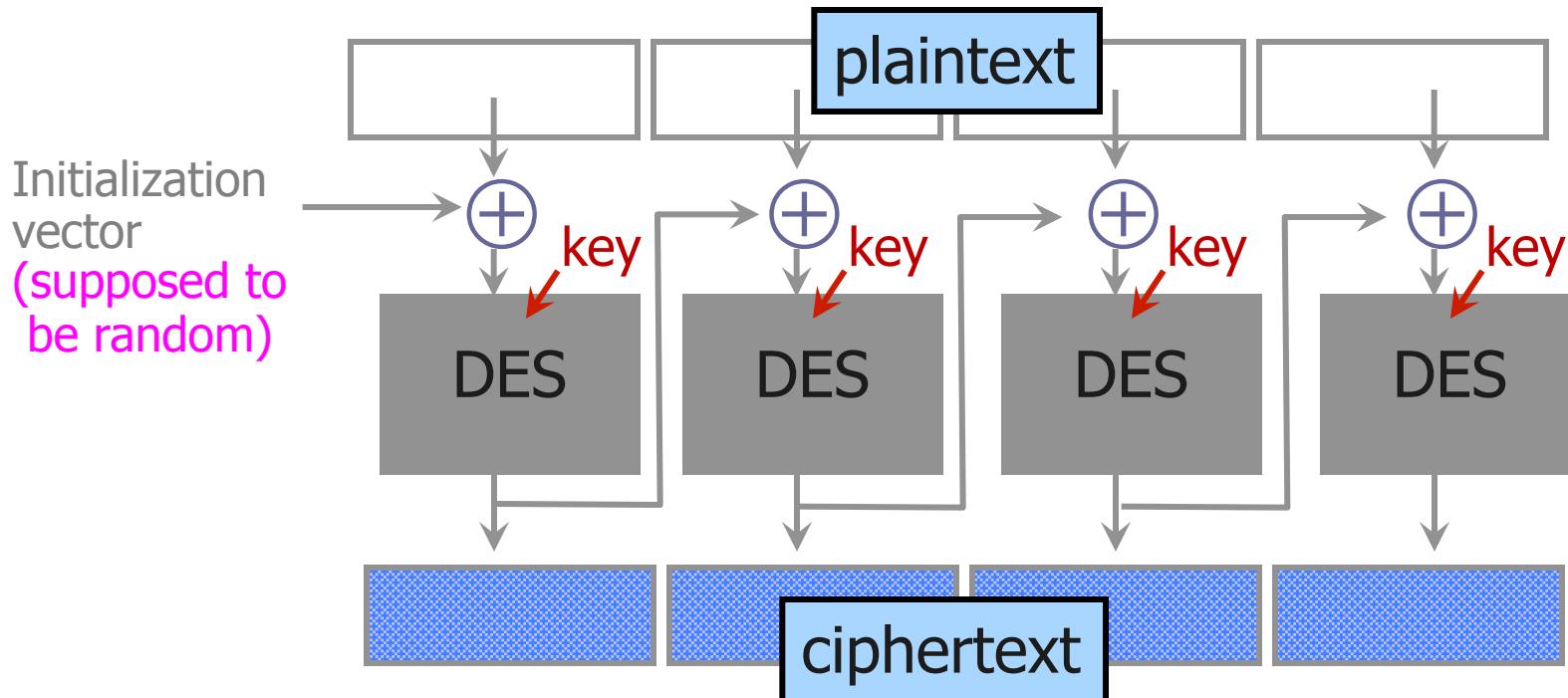
- Best: **fresh, random IV** for every message
  - Can also use unique IV (eg, counter), but then the first step in CBC mode must be  $\text{IV}' \leftarrow E(k, \text{IV})$ 
    - Example: Windows BitLocker
    - May not need to transmit IV with the ciphertext

- ◆ Multi-use key, unique messages

- Synthetic IV:  $\text{IV} \leftarrow F(k', \text{message})$ 
    - F is a cryptographically secure keyed pseudorandom function

# CBC and Electronic Voting

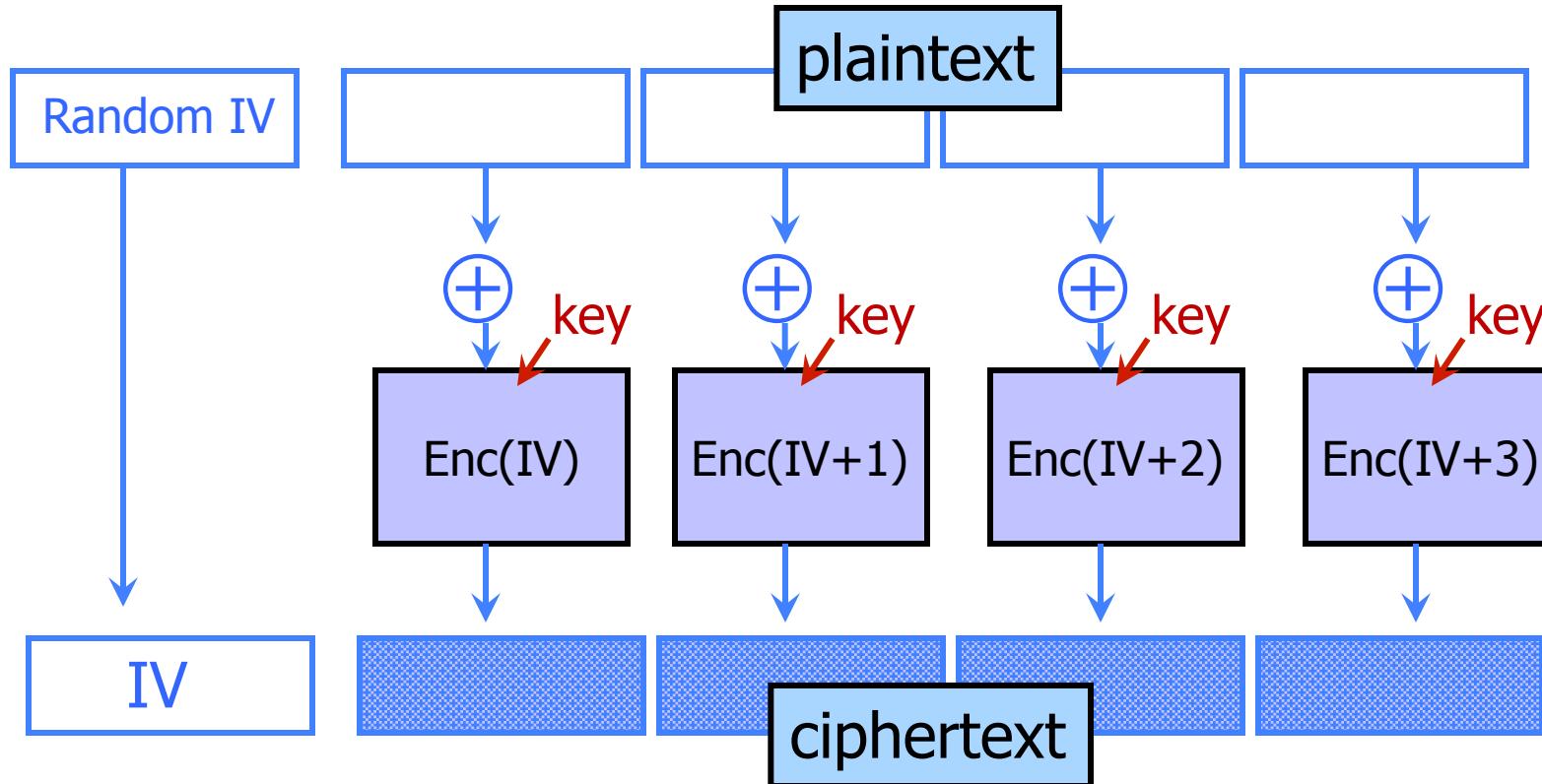
[Kohno, Stubblefield, Rubin, Wallach]



Found in the source code for Diebold voting machines:

```
DesCBCEncrypt( (des_c_block*) tmp, (des_c_block*) record.m_Data,  
    totalSize, DESKEY, NULL, DES_ENCRYPT)
```

# CTR (Counter Mode)



- ◆ Still does not guarantee integrity
- ◆ Fragile if counter repeats

# When Is a Cipher “Secure”?

---

- ◆ Hard to recover plaintext from ciphertext?
  - What if attacker learns only some bits of the plaintext? Some function of the bits? Some partial information about the plaintext?
- ◆ Fixed mapping from plaintexts to ciphertexts?
  - What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?
  - What if attacker guesses the plaintext – can he verify his guess?
  - Implication: encryption must be randomized or stateful

# How Can a Cipher Be Attacked?

- ◆ Attackers knows ciphertext and encryption algthm
  - What else does the attacker know? Depends on the application in which the cipher is used!
- ◆ Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- ◆ Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of his choice
- ◆ Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext except the target
  - Sometimes very realistic



**OpenSSL**  
Cryptography and SSL/TLS Toolkit

# Known-Plaintext Attack

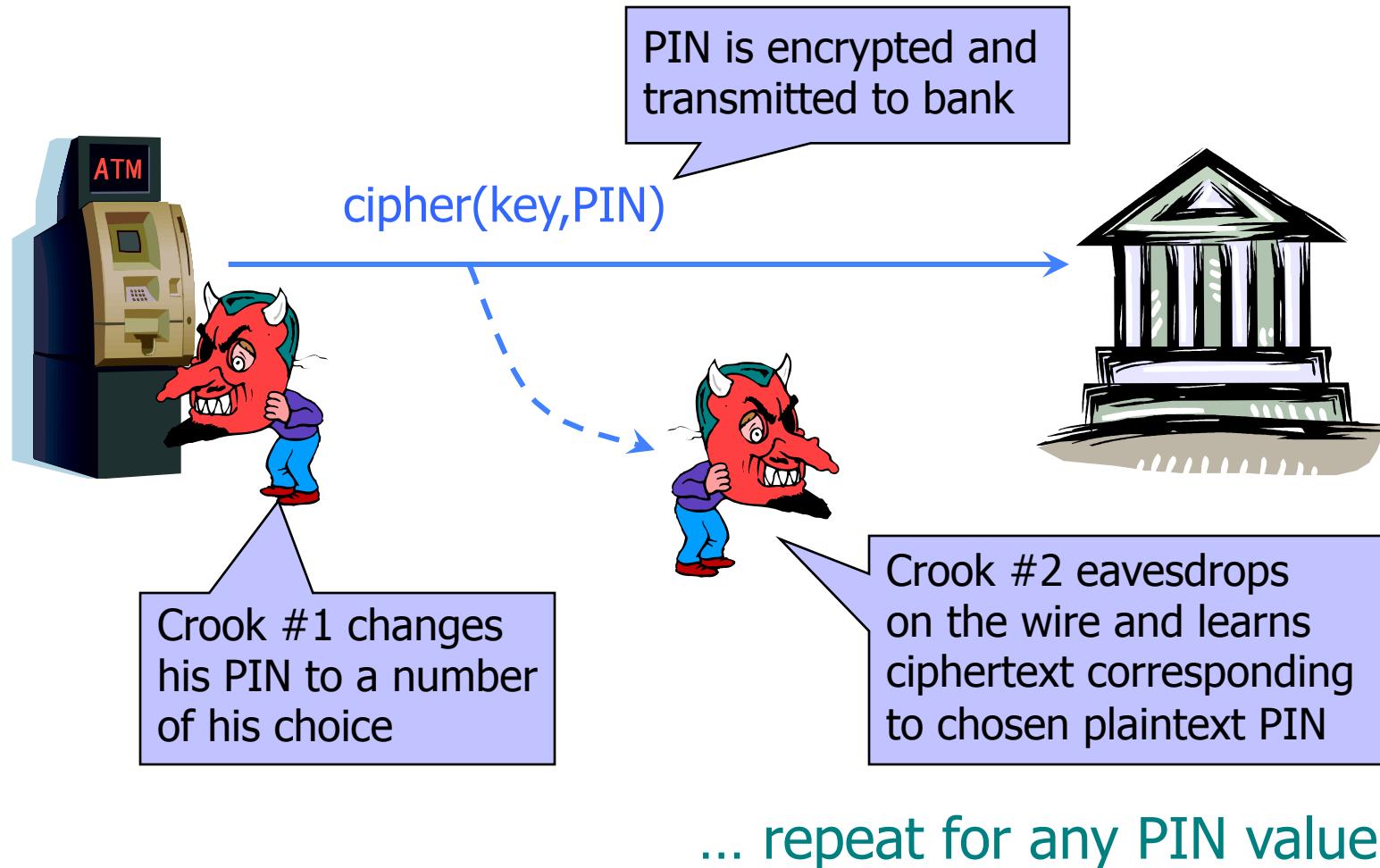
---

[From “The Art of Intrusion”]

Extracting password from an encrypted PKZIP file ...

- ◆ “... I opened the ZIP file and found a ‘logo.tif’ file, so I went to their main Web site and looked at all the files named ‘logo.tif.’ I downloaded them and zipped them all up and found one that matched the same checksum as the one in the protected ZIP file”
- ◆ With known plaintext, PkCrack took 5 minutes to extract the key
  - Biham-Kocher attack on PKZIP stream cipher

# Chosen-Plaintext Attack



# Very Informal Intuition

Minimum security requirement for a modern encryption scheme

- ◆ Security against chosen-plaintext attack
  - Ciphertext leaks no information about the plaintext
  - Even if the attacker correctly guesses the plaintext, he cannot verify his guess
  - Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
- ◆ Security against chosen-ciphertext attack
  - Integrity protection – it is not possible to change the plaintext by modifying the ciphertext

# The Chosen-Plaintext Game

---

- ◆ Attacker does not know the key
- ◆ He chooses as many plaintexts as he wants, and receives the corresponding ciphertexts
- ◆ When ready, he picks two plaintexts  $M_0$  and  $M_1$ 
  - He is even allowed to pick plaintexts for which he previously learned ciphertexts!
- ◆ He receives either a ciphertext of  $M_0$ , or a ciphertext of  $M_1$
- ◆ He wins if he guesses correctly which one it is

# Meaning of “Leaks No Information”

---

- ◆ Idea: given a ciphertext, attacker should not be able to learn **even a single bit** of useful information about the plaintext
- ◆ Let  $\text{Enc}(M_0, M_1, b)$  be a “magic box” that returns encrypted  $M_b$  0 or 1
  - Given two plaintexts, the box always returns the ciphertext of the left plaintext or right plaintext
  - Attacker can use this box to obtain the ciphertext of any plaintext  $M$  by submitting  $M_0=M_1=M$ , or he can try to learn even more by submitting  $M_0 \neq M_1$
- ◆ Attacker’s goal is to learn just this one bit  $b$

# Chosen-Plaintext Security

---

- ◆ Consider two experiments (A is the attacker)

## Experiment 0

A interacts with  $\text{Enc}(-,-,0)$   
and outputs his guess of bit b

## Experiment 1

A interacts with  $\text{Enc}(-,-,1)$   
and outputs his guess of bit b

- Identical except for the value of the secret bit
- b is attacker's guess of the secret bit

- ◆ Attacker's advantage is defined as

$$| \text{Prob}(A \text{ outputs } 1 \text{ in Exp0}) - \text{Prob}(A \text{ outputs } 1 \text{ in Exp1}) |$$

- ◆ Encryption scheme is chosen-plaintext secure if this advantage is negligible for any efficient A

# Simple Example

---

- ◆ Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with  $\text{Enc}(\cdot, \cdot, b)$

Let  $X, Y$  be any two different plaintexts

$C_1 \leftarrow \text{Enc}(X, X, b); \quad C_2 \leftarrow \text{Enc}(X, Y, b);$

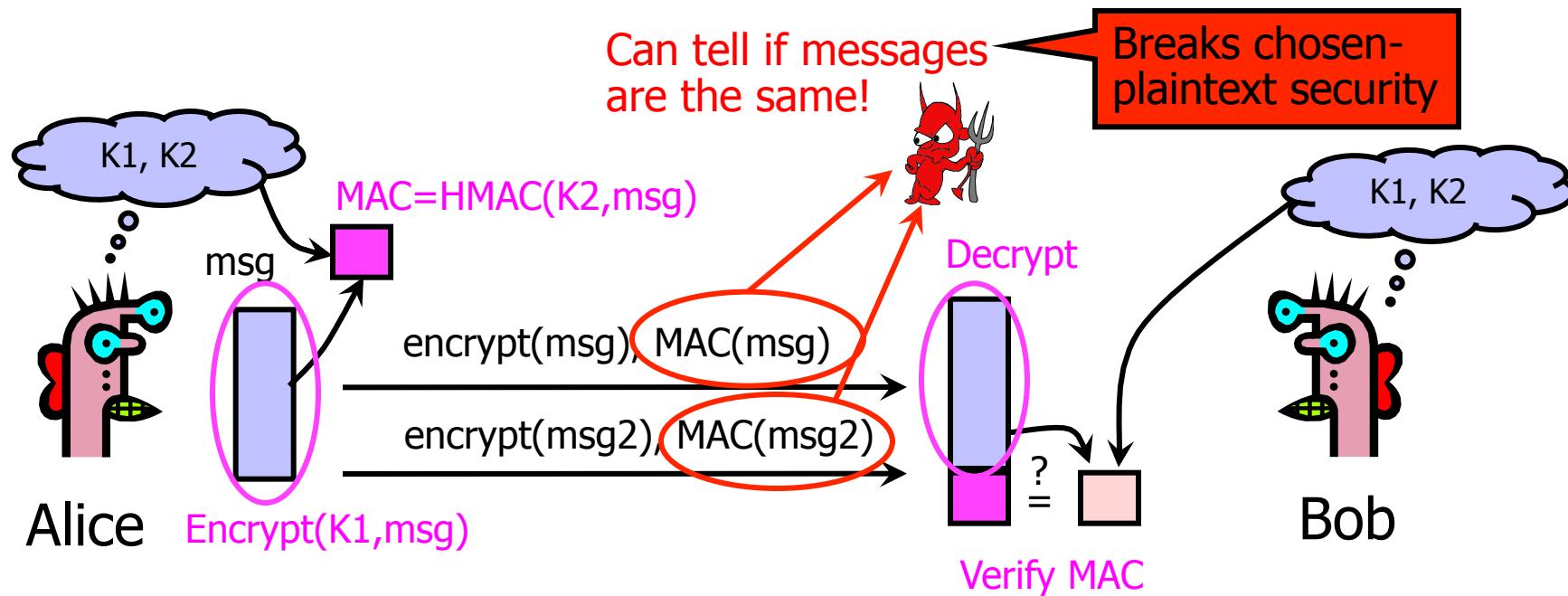
If  $C_1 = C_2$  then  $b=0$  else  $b=1$

- ◆ The advantage of this attacker A is 1

$\text{Prob}(A \text{ outputs 1 if } b=0) = 0 \quad \text{Prob}(A \text{ outputs 1 if } b=1) = 1$

# Encrypt + MAC

Goal: confidentiality + integrity + authentication



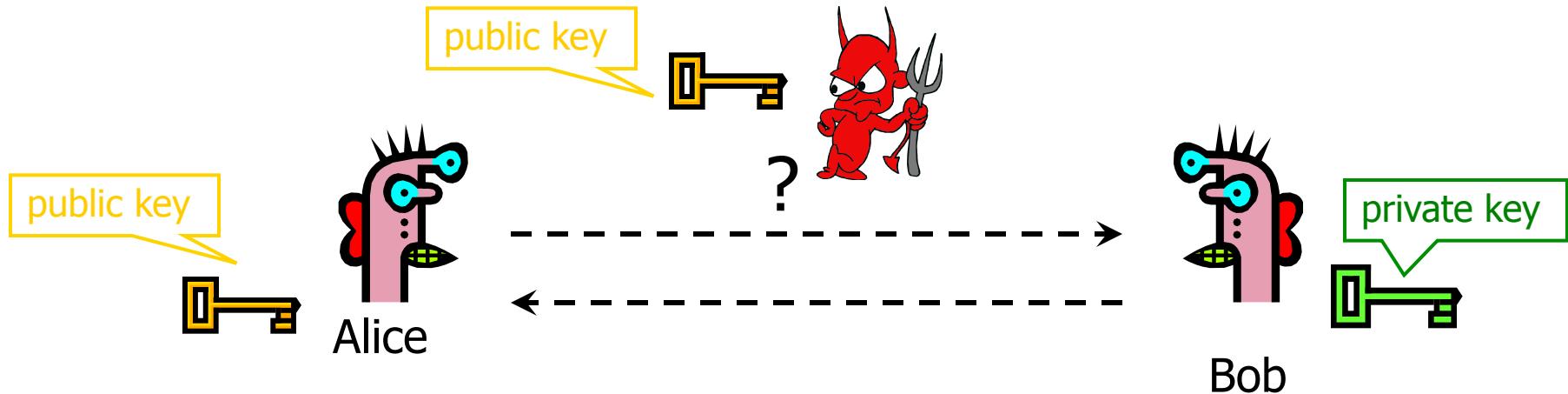
MAC is deterministic: messages are equal  $\Rightarrow$  their MACs are equal

**Solution:** Encrypt, then MAC (or MAC, then encrypt)

# Overview of Public-Key Cryptography

---

# Public-Key Cryptography



Given: Everybody knows Bob's **public key**

- How is this achieved in practice?

Only Bob knows the corresponding **private key**

- Goals:
1. Alice wants to send a message that only Bob can read
  2. Bob wants to send a message that only Bob could have written

# Applications of Public-Key Crypto

---

## ◆ Encryption for confidentiality

- Anyone can encrypt a message
  - With symmetric crypto, must know the secret key to encrypt
- Only someone who knows the private key can decrypt
- Secret keys are only stored in one place

## ◆ Digital signatures for authentication

- Only someone who knows the private key can sign

## ◆ Session key establishment

- Exchange messages to create a secret session key
- Then switch to symmetric cryptography (why?)

# Public-Key Encryption

---

- ◆ **Key generation:** computationally easy to generate a pair (public key PK, private key SK)
- ◆ **Encryption:** given plaintext M and public key PK, easy to compute ciphertext  $C=E_{PK}(M)$
- ◆ **Decryption:** given ciphertext  $C=E_{PK}(M)$  and private key SK, easy to compute plaintext M
  - Infeasible to learn anything about M from C without SK
  - Trapdoor function:  $\text{Decrypt}(\text{SK}, \text{Encrypt}(\text{PK}, \text{M})) = \text{M}$

3  
func $\downarrow$

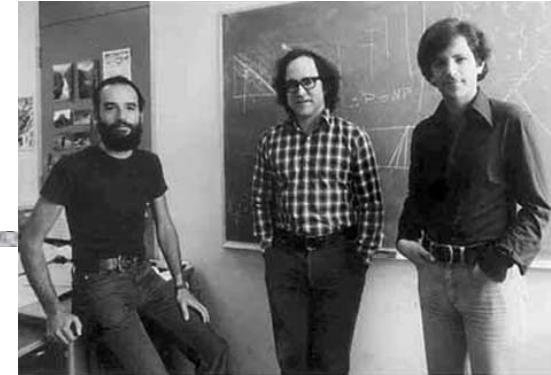
# Some Number Theory Facts

- ◆ Euler totient function  $\varphi(n)$  where  $n \geq 1$  is the number of integers in the  $[1, n]$  interval that are relatively prime to  $n$ 
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- ◆ Euler's theorem:  
if  $a \in \mathbb{Z}_n^*$ , then  $a^{\varphi(n)} \equiv 1 \pmod{n}$
- ◆ Special case: Fermat's Little Theorem  
if  $p$  is prime and  $\gcd(a, p) = 1$ , then  $a^{p-1} \equiv 1 \pmod{p}$



$\uparrow$   
 $n$

# RSA Cryptosystem



## ◆ Key generation:

- Generate large primes  $p, q$   
– At least 2048 bits each... need primality testing!
- Compute  $n=pq$  *must be prime to n (not divisible by)*  
– Note that  $\varphi(n) = (p-1)(q-1)$  think about  $(p-1)(q-1)$   $5 \times 7 = 35$   
 $4 \times 6 = 24$
- Choose small  $e$ , relatively prime to  $\varphi(n)$   
– Typically,  $e=3$  (may be vulnerable) or  $e=2^{16}+1=65537$  (why?)
- Compute unique  $d$  such that  $ed \equiv 1 \pmod{\varphi(n)}$  *larger  
↓  
more secure  
but expensive  
to compute.*
- Public key =  $(e, n)$ ; private key =  $d$

## ◆ Encryption of $m$ : $c = m^e \pmod{n}$

## ◆ Decryption of $c$ : $c^d \pmod{n} = (m^e)^d \pmod{n} = m$

# Why RSA Decryption Works

- ◆  $e \cdot d \equiv 1 \pmod{\varphi(n)}$
- ◆ Thus  $e \cdot d = 1 + k \cdot \varphi(n) = 1 + k(p-1)(q-1)$  for some  $k$
- ◆ If  $\gcd(m, p) = 1$ , then by Fermat's Little Theorem,  
 $m^{p-1} \equiv 1 \pmod{p}$
- ◆ Raise both sides to the power  $k(q-1)$  and multiply by  $m$ , obtaining  $m^{\underline{1+k(p-1)(q-1)}} \equiv \underline{m \pmod{p}}$
- ◆ Thus  $m^{ed} \equiv m \pmod{p}$
- ◆ By the same argument,  $m^{ed} \equiv m \pmod{q}$
- ◆ Since  $p$  and  $q$  are distinct primes and  $p \cdot q = n$ ,  
 $m^{ed} \equiv m \pmod{n}$

$$n = pq$$

$$(m^{p-1})^{\frac{k(q-1)}{}} \equiv 1 \pmod{p} \equiv m^{\frac{kp-1+q-1}{}} \equiv m \pmod{p}$$

$$m^{\overset{ed}{\underset{,}{}}} \equiv m \pmod{p}$$

# Why Is RSA Secure?

- ◆ **RSA problem:** given  $c$ ,  $n=pq$ , and  $e$  such that  $\gcd(e, (p-1)(q-1))=1$ , find  $m$  such that  $m^e \equiv c \pmod{n}$

$$\begin{aligned} & c, n, e \\ & \rightarrow \gcd(e, \frac{(p-1)(q-1)}{n}) = 1 \end{aligned}$$

- In other words, recover  $m$  from ciphertext  $c$  and public key  $(n,e)$  by taking  $e^{\text{th}}$  root of  $c$  modulo  $n$
- There is no known efficient algorithm for doing this
- ◆ **Factoring problem:** given positive integer  $n$ , find primes  $p_1, \dots, p_k$  such that  $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$
- ◆ If factoring is easy, then RSA problem is easy, but may be possible to break RSA without factoring  $n$

# “Textbook” RSA Is Bad Encryption

---

- ◆ Deterministic (*Not random*)

- Attacker can guess plaintext, compute ciphertext, and compare for equality
- If messages are from a small set (for example, yes/no), can build a table of corresponding ciphertexts

- ◆ Can tamper with encrypted messages

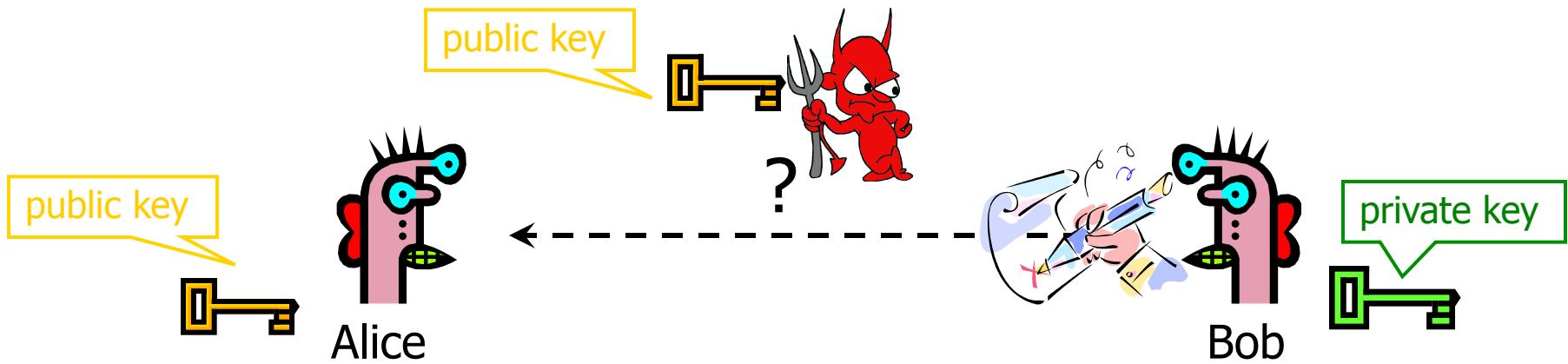
- Take an encrypted auction bid  $c$  and submit  $c(101/100)^e \text{ mod } n$  instead

- ◆ Does not provide **semantic security** (security against chosen-plaintext attacks)

# Integrity in RSA Encryption

- ◆ “Textbook” RSA does not provide integrity
  - Given encryptions of  $m_1$  and  $m_2$ , attacker can create encryption of  $m_1 \cdot m_2$  *not beware of chg* *modifies ciphertext*  
$$(m_1^e) \cdot (m_2^e) \text{ mod } n \equiv (m_1 \cdot m_2)^e \text{ mod } n$$
  - Attacker can convert  $m$  into  $m^k$  without decrypting  
$$(m^e)^k \text{ mod } n \equiv (m^k)^e \text{ mod } n$$
 *how? public key (e, n)*
- ◆ In practice, OAEP is used: instead of encrypting  $M$ , encrypt  $M \oplus G(r)$  ;  $r \oplus H(M \oplus G(r))$ 
  - $r$  is random and fresh,  $G$  and  $H$  are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - ... if hash functions are “good” and RSA problem is hard  
*Collision free*

# Digital Signatures: Basic Idea



Given: Everybody knows Bob's **public key**  
Only Bob knows the corresponding **private key**

Goal: Bob sends a “digitally signed” message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

# RSA Signatures

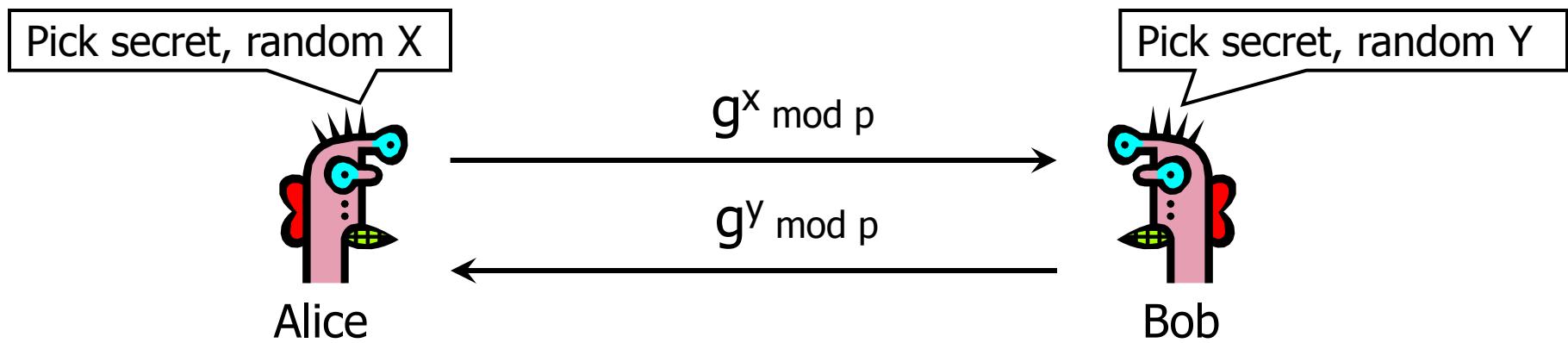
---

- ◆ Public key is  $(n, e)$ , private key is  $d$
- ◆ To sign message  $m$ :  $s = \text{hash}(m)^d \bmod n$ 
  - Signing and decryption are the same mathematical operation in RSA
- ◆ To verify signature  $s$  on message  $m$ :  
$$s^e \bmod n = (\text{hash}(m)^d)^e \bmod n = \text{hash}(m)$$
  - Verification and encryption are the same mathematical operation in RSA
- ◆ Message must be hashed and padded (why?)  
*Attack is ez to alter*

# Diffie-Hellman Protocol



- ◆ Alice and Bob never met and share no secrets
- ◆ Public info:  $p$  and  $g$ 
  - $p$  is a large prime number,  $g$  is a generator of  $\mathbb{Z}_p^*$ 
    - $\mathbb{Z}_p^* = \{1, 2 \dots p-1\}; \forall a \in \mathbb{Z}_p^* \exists i \text{ such that } a = g^i \bmod p$



Compute  $k = (g^y)^x = g^{xy} \bmod p$

Compute  $k = (g^x)^y = g^{xy} \bmod p$

# Why Is Diffie-Hellman Secure?

---

- ◆ Discrete Logarithm (DL) problem:  
given  $g^x \bmod p$ , it's hard to extract  $x$ 
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- ◆ Computational Diffie-Hellman (CDH) problem:  
given  $g^x$  and  $g^y$ , it's hard to compute  $g^{xy} \bmod p$ 
  - ... unless you know  $x$  or  $y$ , in which case it's easy
- ◆ Decisional Diffie-Hellman (DDH) problem:  
given  $g^x$  and  $g^y$ , it's hard to tell the difference between  $g^{xy} \bmod p$  and  $g^r \bmod p$  where  $r$  is random

# Properties of Diffie-Hellman

---

- ◆ Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between the established key and a random value
  - Can use the new key for symmetric cryptography
- ◆ Basic Diffie-Hellman protocol does not provide authentication
  - IPsec combines Diffie-Hellman with signatures, anti-DoS cookies, etc.

# Advantages of Public-Key Crypto

---

- ◆ Confidentiality without shared secrets
  - Very useful in open environments
  - Can use this for key establishment, avoiding the “chicken-or-egg” problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
- ◆ Authentication without shared secrets
- ◆ Encryption keys are public, but must be sure that Alice's public key is really her public key
  - This is a hard problem... Often solved using public-key certificates

# Disadvantages of Public-Key Crypto

---

- ◆ Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - SSL, IPsec, most other systems based on public crypto
- ◆ Keys are longer
  - 2048 bits (RSA) rather than 128 bits (AES)
- ◆ Relies on unproven number-theoretic assumptions
  - Factoring, RSA problem, discrete logarithm problem, decisional Diffie-Hellman problem...