

# Overview of Symmetric Encryption

---

Vitaly Shmatikov

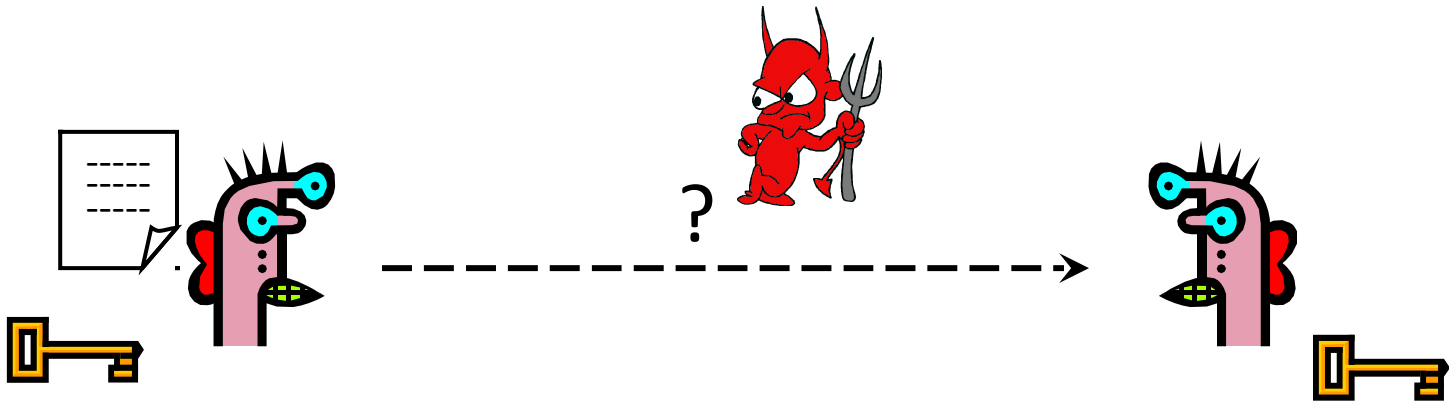
# Reading Assignment

---

- ◆ Read Kaufman 2.1-4 and 4.2

# 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.

### Readers' Comments


Readers shared their thoughts on this article.


[Read All Comments \(127\) »](#)


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

 Uptime

IT and business computing insights

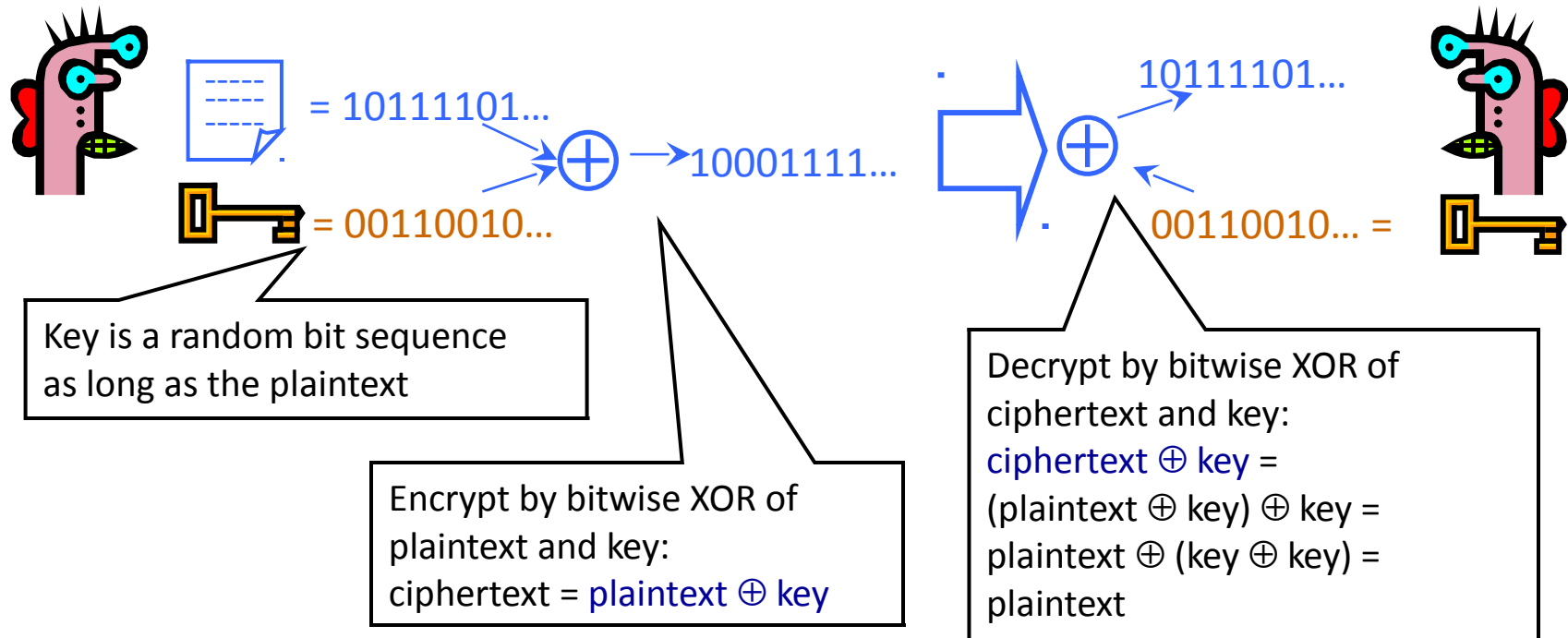


Internet users, there is nothing a sites will need to make changes to said

**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)

# Diagram Transcription

---

Sender has a bit-stream message and a bit-stream key of the same length. The ciphertext is defined as  $\text{message}[i] \text{ XOR } \text{key}[i]$  for every  $i$  in the message. The recipient can decrypt the message by performing the identical operation but using the ciphertext instead of the message, because:

$$\begin{aligned} & \text{ciphertext xor key} \\ &= (\text{plaintext xor key}) \text{ xor key} \\ &= \text{plaintext xor } (\text{key xor key}) \\ &= \text{plaintext xor } (00\dots00) = \text{plaintext} \end{aligned}$$

# 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?

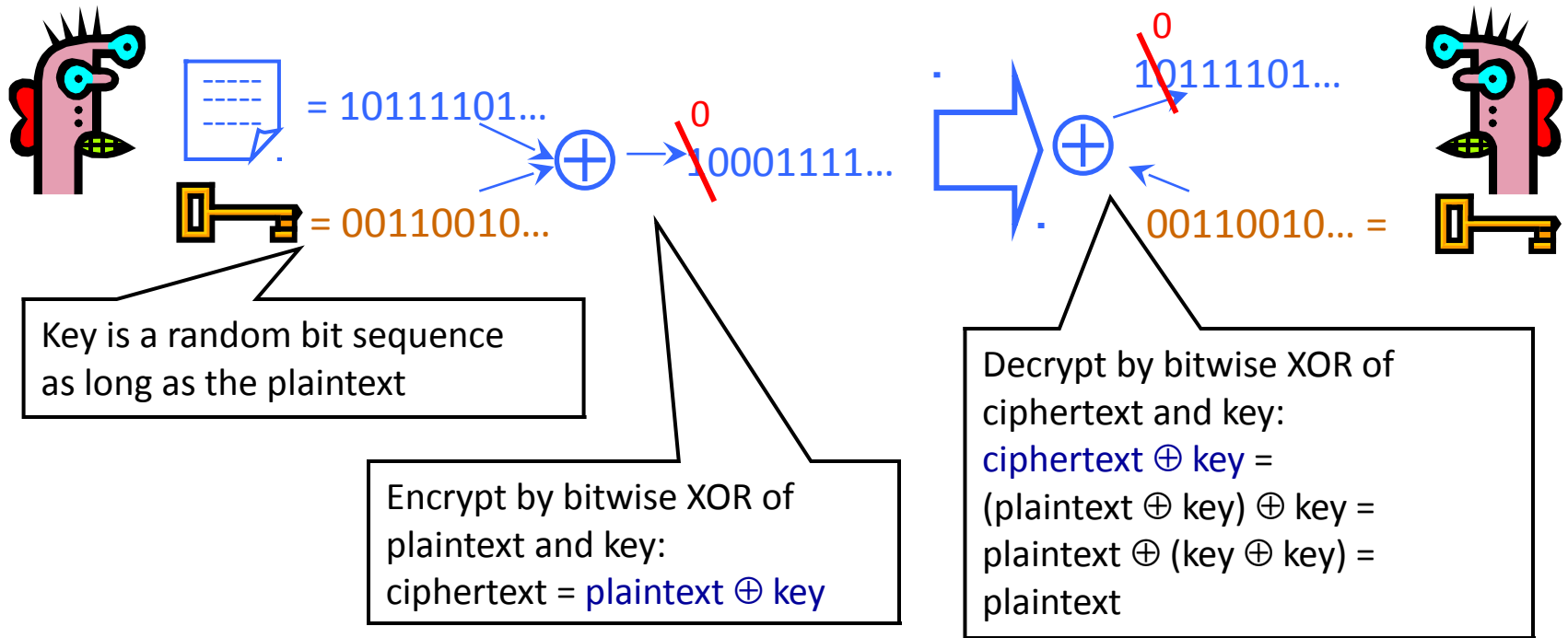


# Problems with One-Time Pad

---

- ◆ 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



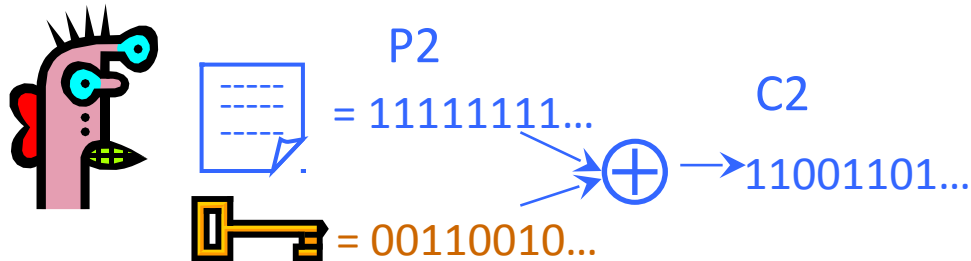
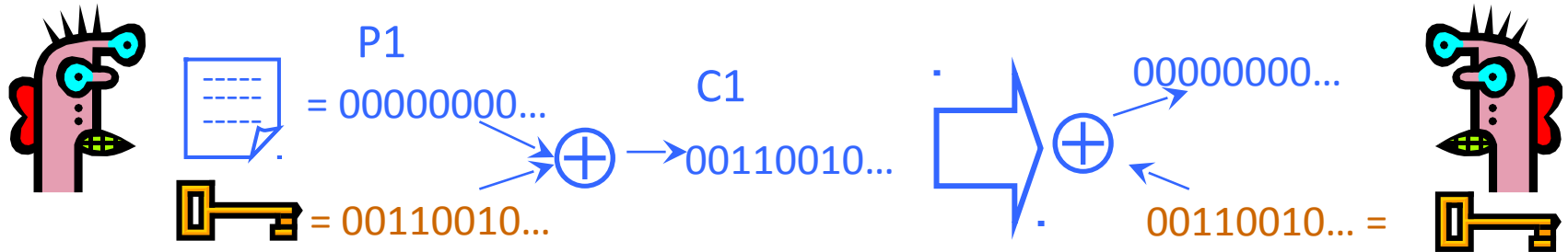
# Diagram transcription

---

No integrity is provided, because if the attacker flips bit  $n$  of the ciphertext while it is in transit, then after being decrypted bit  $n$  of the plaintext will be flipped.

That is to say that while the attacker cannot know the contents of the message, he or she can choose to flip any bits from their original values.

# Dangers of Reuse



Learn relationship between plaintexts

$$C1 \oplus C2 = (P1 \oplus K) \oplus (P2 \oplus K) =$$

$$(P1 \oplus P2) \oplus (K \oplus K) = P1 \oplus P2$$



# Diagram Transcription

---

If two ciphertexts are generated using the same key, and the attacker gains access to the ciphertexts (and knows that they were generated with the same key), then:

$$\text{ciphertext1} = \text{plaintext1} \text{ xor } \text{key}$$

$$\text{ciphertext2} = \text{plaintext2} \text{ xor } \text{key}$$

So

$$\text{ciphertext1} \text{ xor } \text{ciphertext2}$$

$$= (\text{plaintext1} \text{ xor } \text{key}) \text{ xor } (\text{plaintext2} \text{ xor } \text{key})$$

$$= (\text{plaintext1} \text{ xor } \text{plaintext2}) \text{ xor } (\text{key} \text{ xor } \text{key})$$

$$= (\text{plaintext1} \text{ xor } \text{plaintext2})$$

# 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

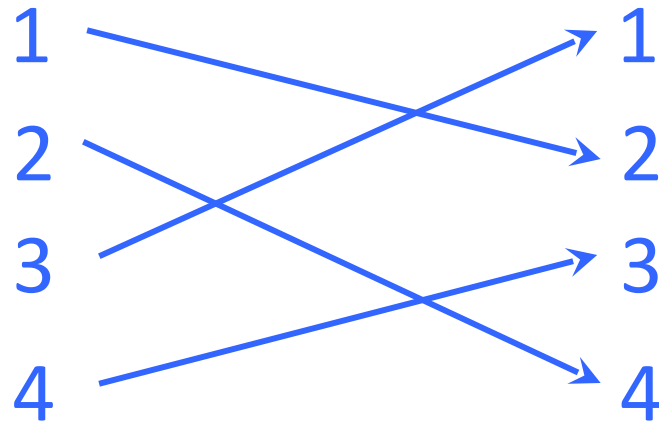
# Block Ciphers

---

- ◆ 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

---



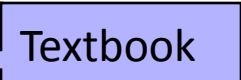

CODE becomes DCEO

- ◆ 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”

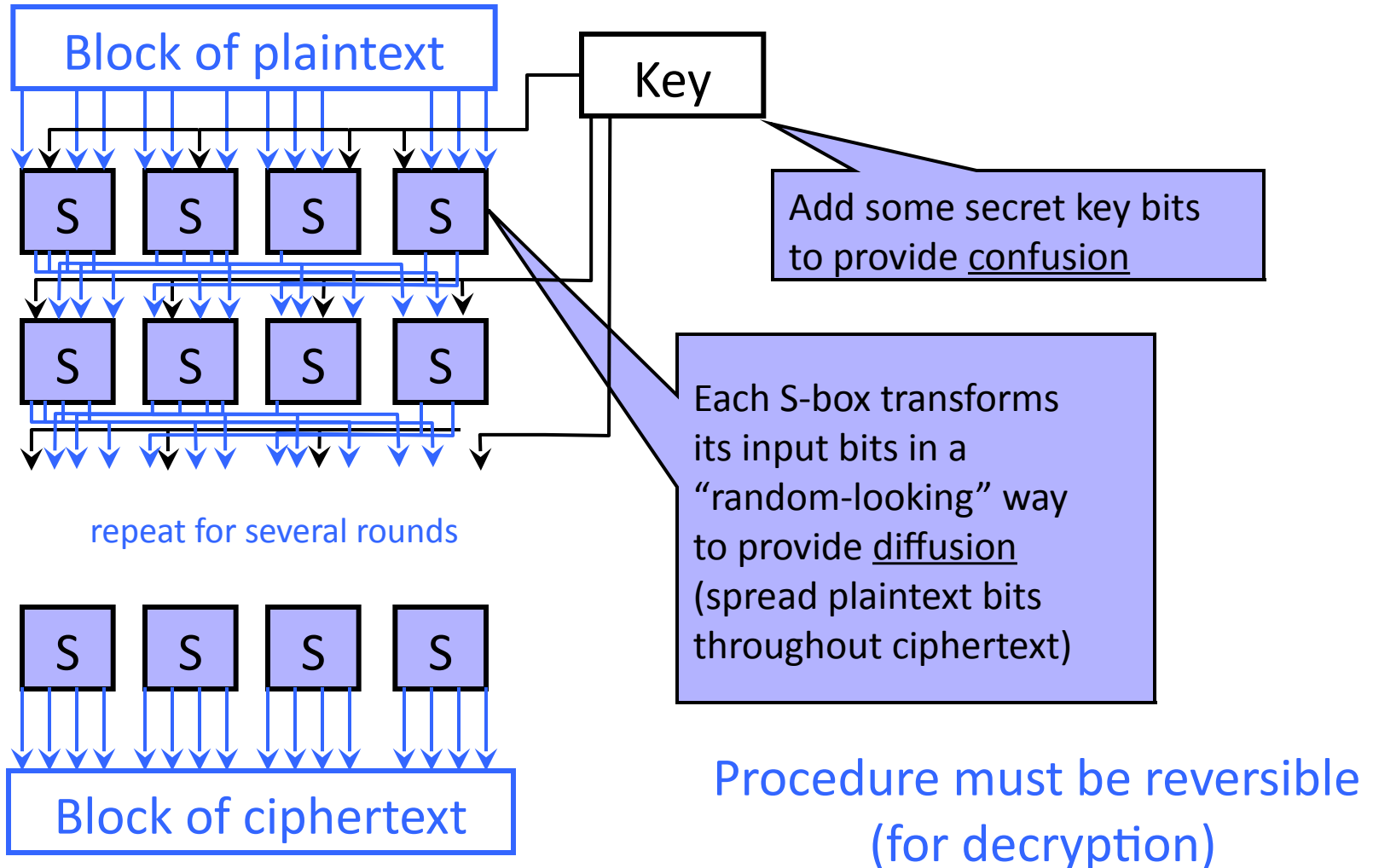


# A Bit of Block Cipher History

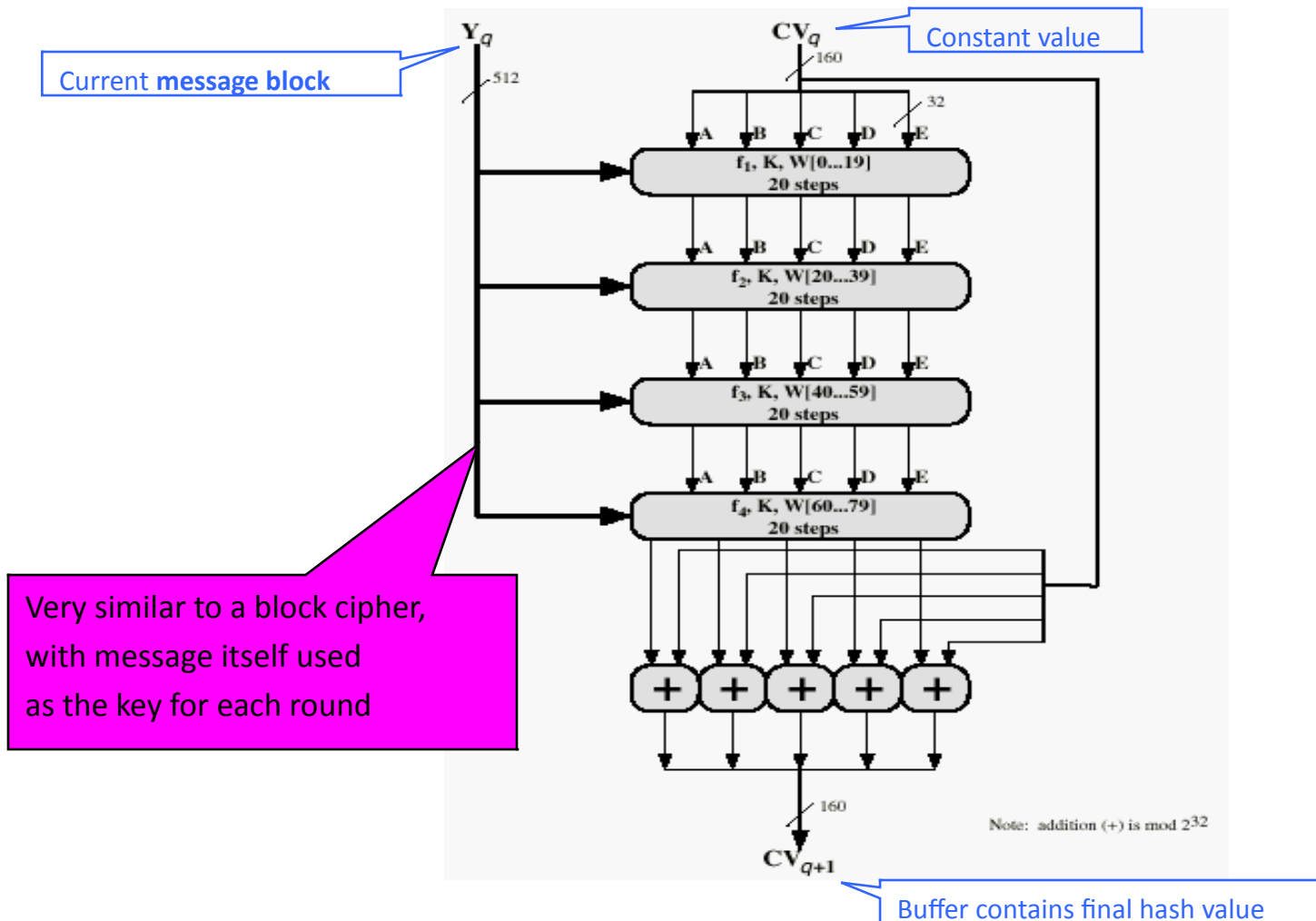
---

- ◆ Playfair and variants (from 1854 until WWII)
- ◆ Feistel structure 
  - “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 
  - 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?

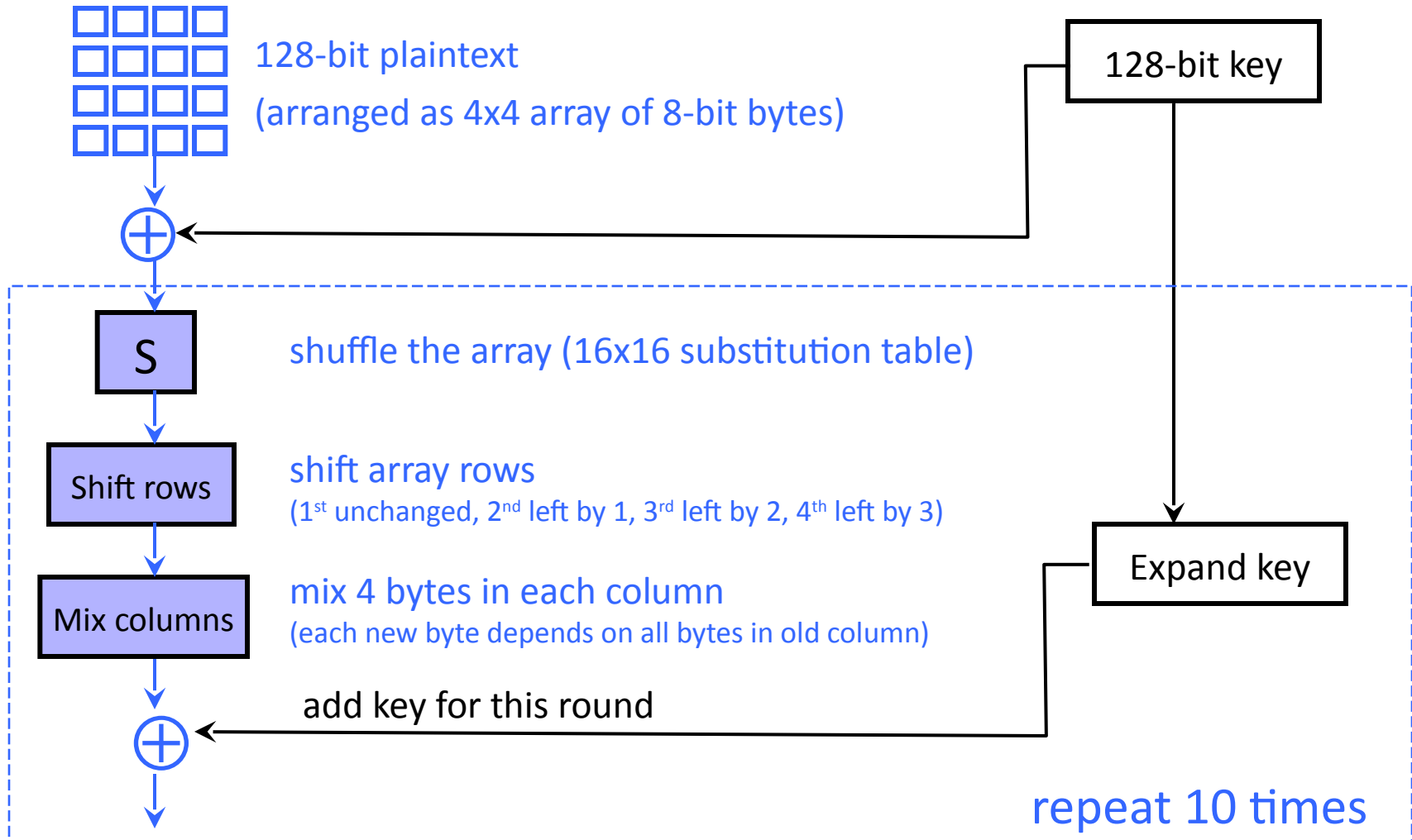


# Advanced Encryption Standard (AES)

---

- ◆ 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



# Diagram Transcription

---

The basic structure of Rijndael is to take the 128-bit plaintext, interpret it as a 4x4 matrix of 8-bit bytes, and then:

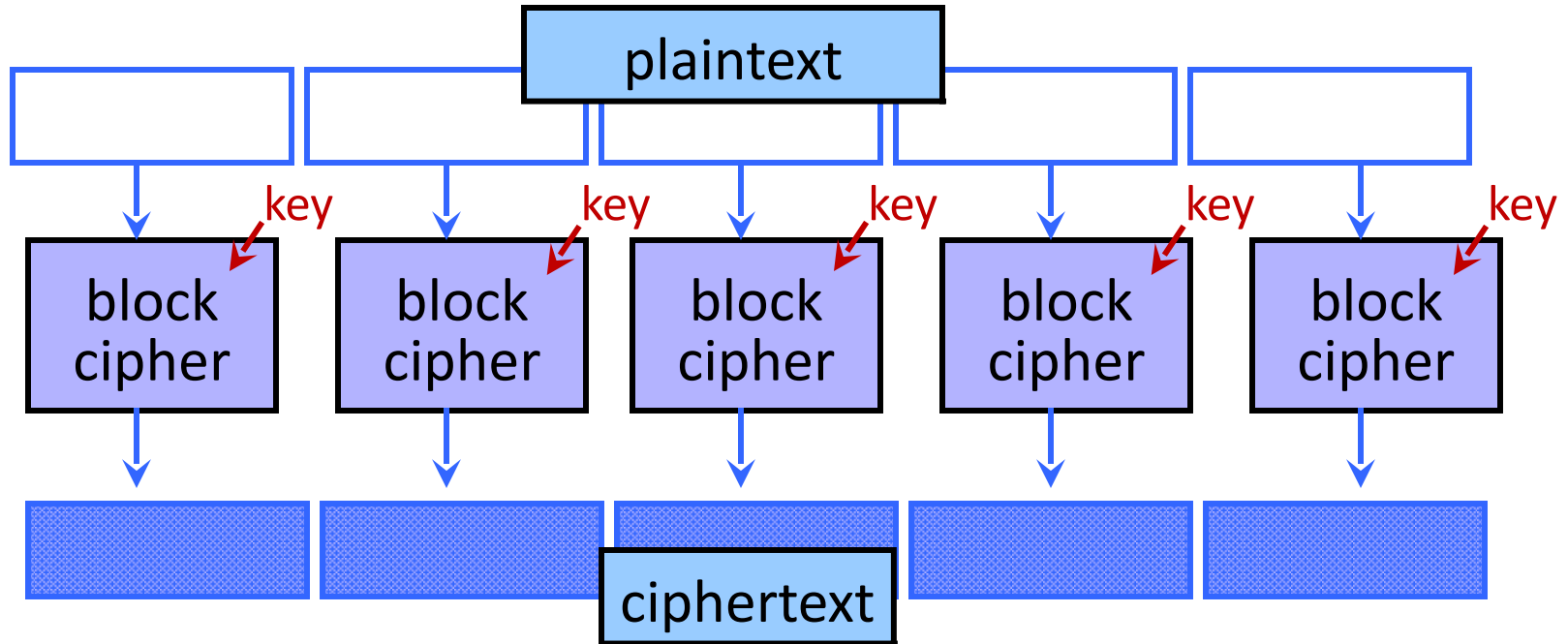
- shuffle the array according to a substitution table
- shift the rows
- mix the 4-byte columns
- mix in key material
- repeat 10 times

# 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.

# ECB Mode



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



# Diagram Transcription

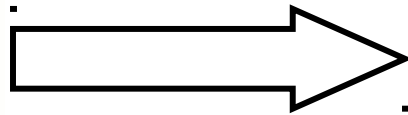
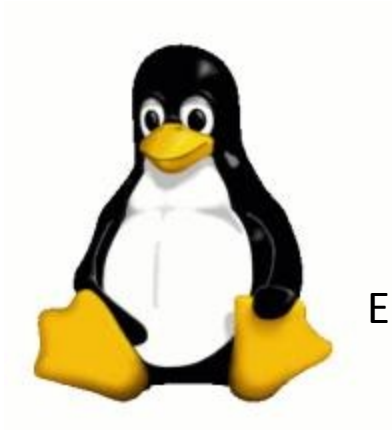
---

The plaintext is divided into blocks, and each block is individually encrypted using the key. The ciphertext is the output blocks concatenated with each other.

(contains diagram, no transcription [visual example])

# Information Leakage in ECB Mode

[Wikipedia]



Encrypt in ECB mode



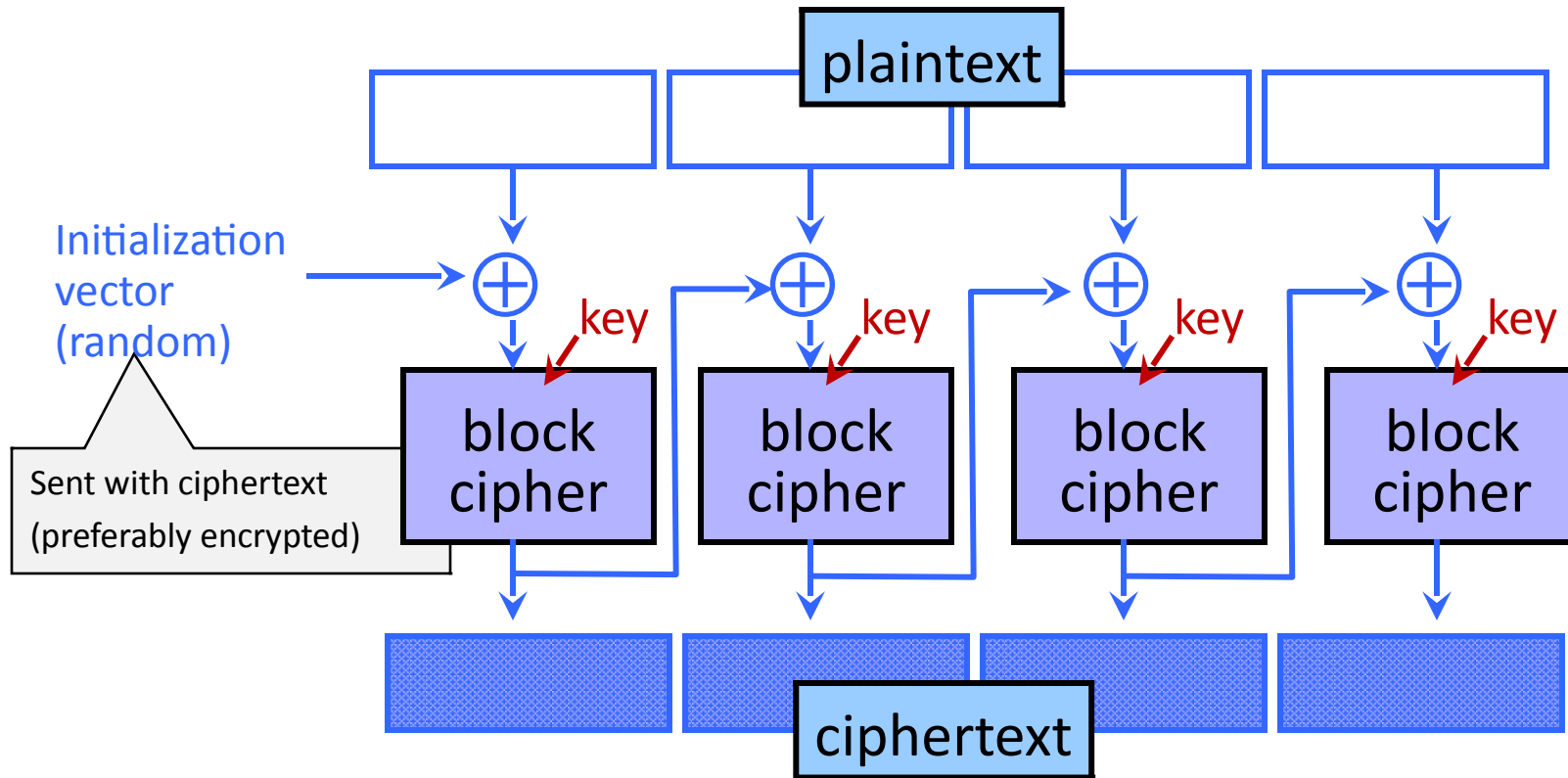
# Adobe Passwords Stolen (2013)

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

```
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-|-adIewKvmJEsFqx0HFoFrXg==|-|--
96649467-|--|-.ius@ic.fbi.gov-|-lsYw5KRKNT/ioxG6CatHBw==|-glass o
96670195-|--|-.fbi.gov-|-X4+k4uhyDh/ioxG6CatHBw==|-|--
105095956-|--|-.earthlink.net-|-ZU2tTTFIZq/ioxG6CatHBw==|-socialsecurity#|--
108260815-|--|-.r@genext.net-|-MuKnZ7KtsiHioxG6CatHBw==|-socialsecurity|--
83508352-|--|-.h@hotmail.com-|-ADEcoaN2oUM=-|-socialsecurityno.--
83023162-|--|-.k390@aol.com-|-9HT+kVHQfs4=-|-socialsecurity name|--
90331688-|--|-.b.edu-|-nNiWecoZTBmXrIXpAZiRHQ==|-ssn#|--
]
```

Password hints

# CBC Mode: Encryption



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

# Diagram Transcription

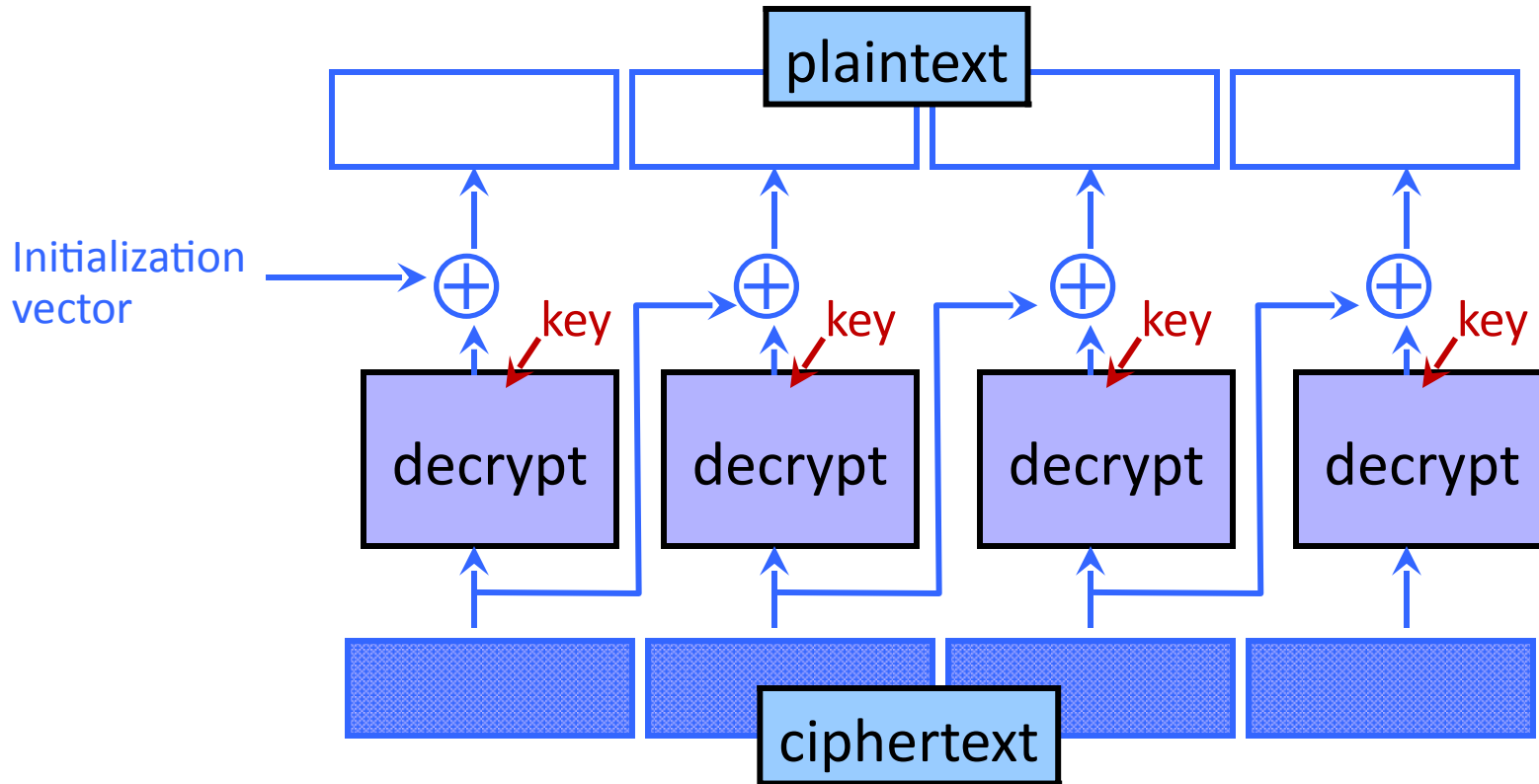
---

The first block of the plaintext is XOR'ed with the IV (random data), before being encrypted. The result of this encryption is the first block of the ciphertext, and then is XOR'ed with the plaintext of the second block before being encrypted, etc.

The IV is pre-pended to the ciphertext (as it is necessary for decryption)

(contains diagram)

# CBC Mode: Decryption



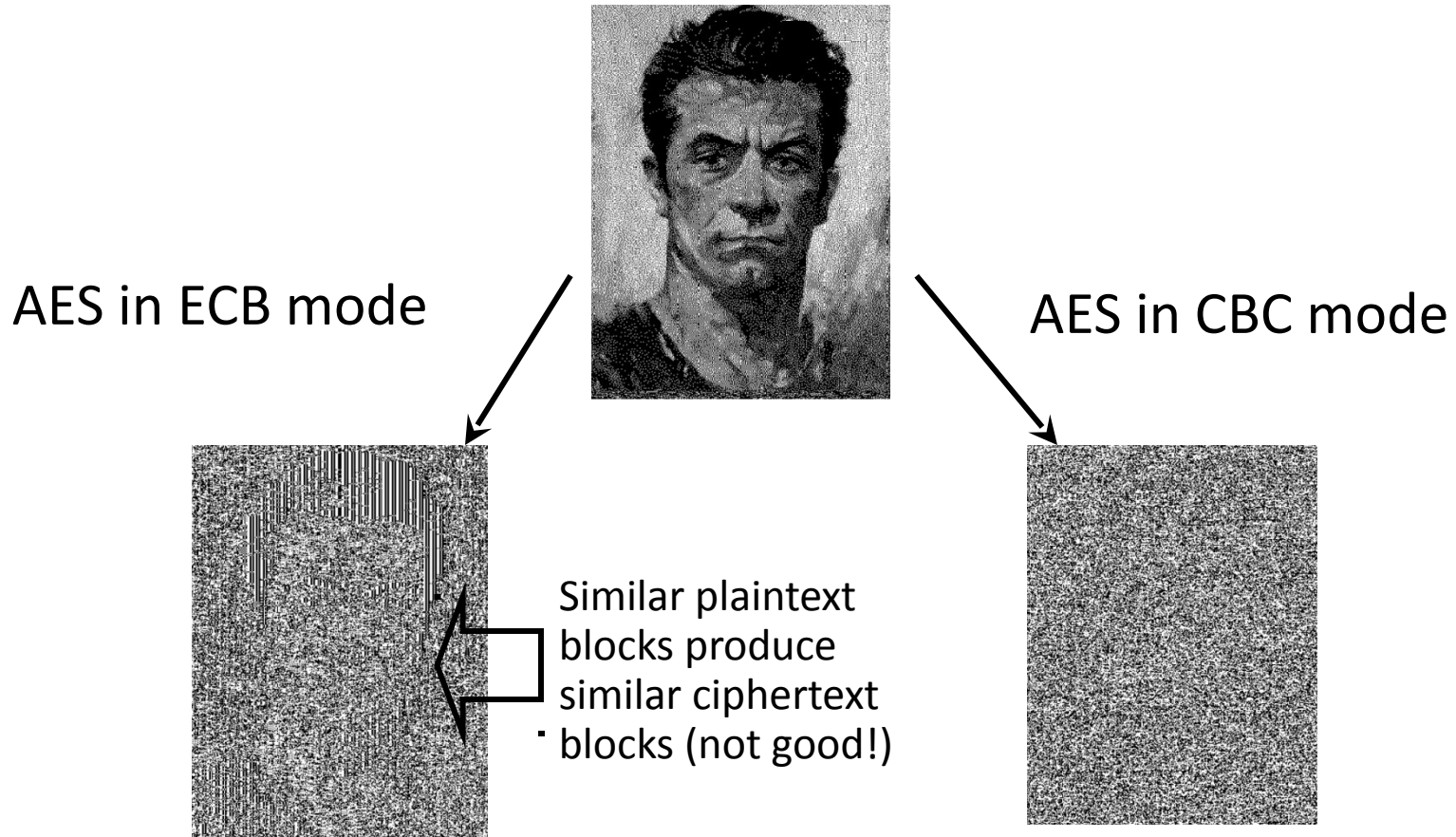
# Diagram Transcription

---

To decrypt in CBC mode, take the first block of the ciphertext (this is the IV). Then decrypt the second block of the ciphertext using the key, and XOR the result as the IV. This is the first block of plaintext. Then take the third block of the ciphertext, decrypt it using the key, and XOR the result with the second block of the ciphertext. This is the second block of plaintext. Rinse and repeat.

# ECB vs. CBC

[Picture due to Bart Preneel]





# Choosing the Initialization Vector

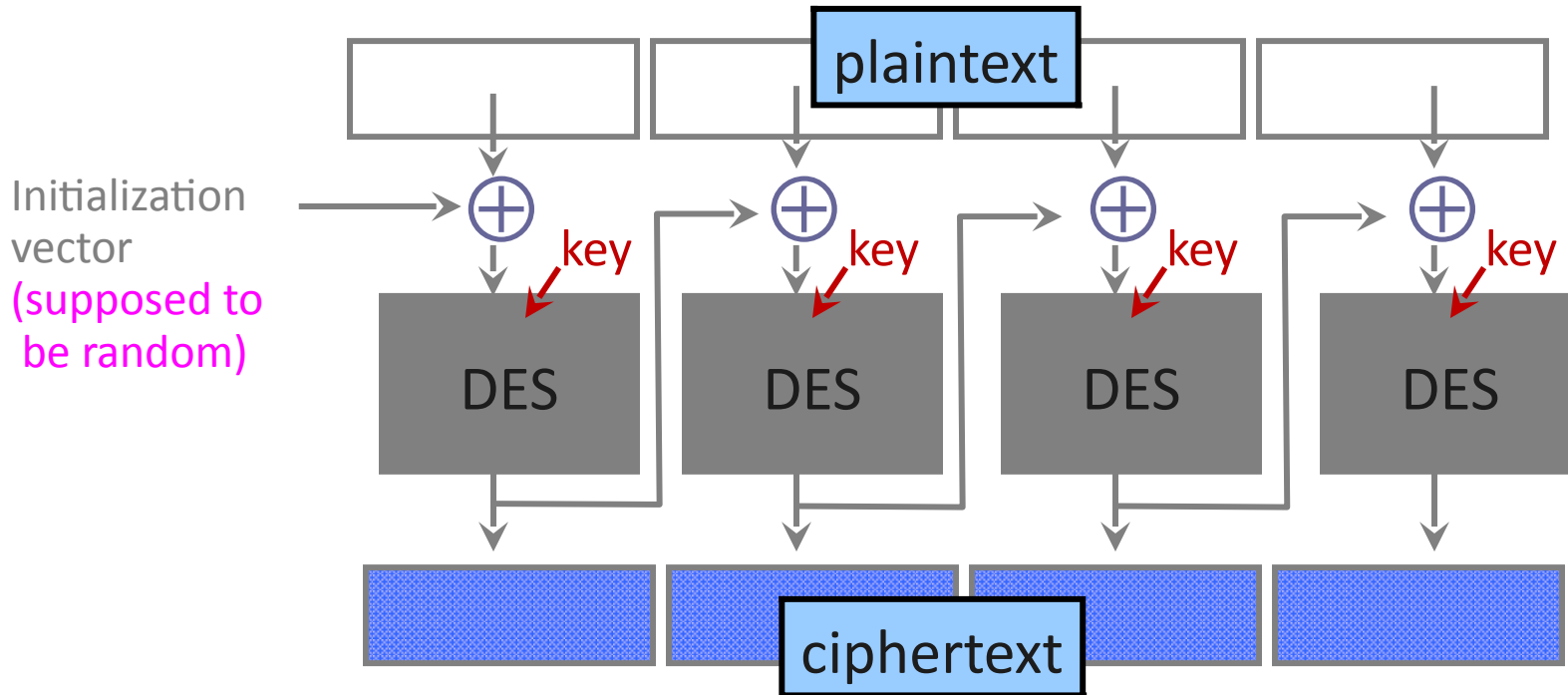
---

- ◆ 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  $IV' \leftarrow E(k, IV)$ 
    - Example: Windows BitLocker
    - May not need to transmit IV with the ciphertext
- ◆ Multi-use key, unique messages
  - Synthetic IV:  $IV \leftarrow F(k', \text{message})$ 
    - F is a cryptographically secure keyed pseudorandom function

(contains diagram, no transcription, see earlier slide on CBC)

# CBC and Electronic Voting

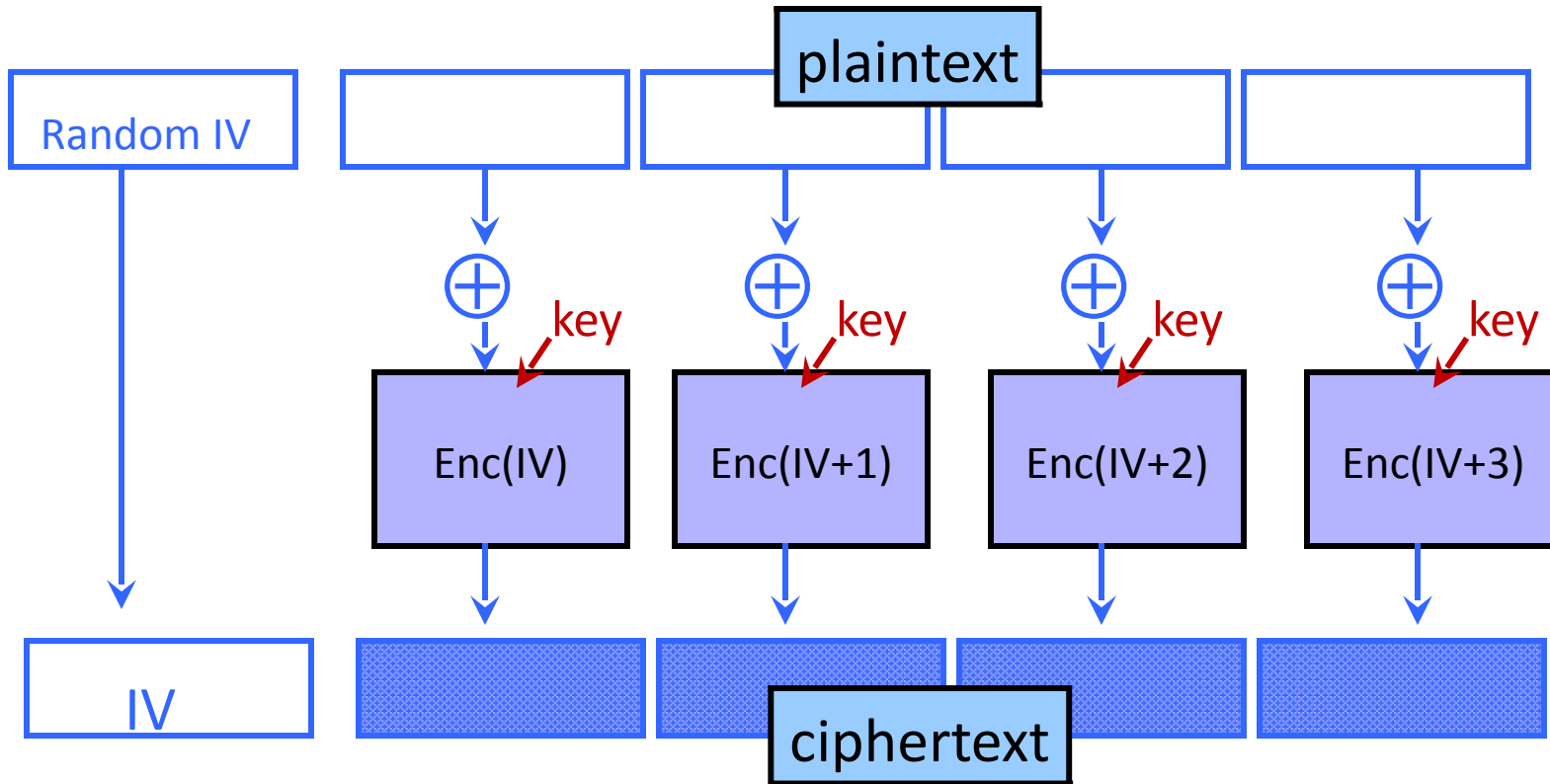
[Kohno, Stubblefield, Rubin, Wallach]



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

# Diagram Transcription

---

Instead of XORing with the IV and then the previous block of ciphertext, first XOR with the IV and then IV+1 and then IV+2 etc etc. Since this XORing happens before encryption, the resulting blocks of ciphertext will be arbitrarily different.

# 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 alghm
  - 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



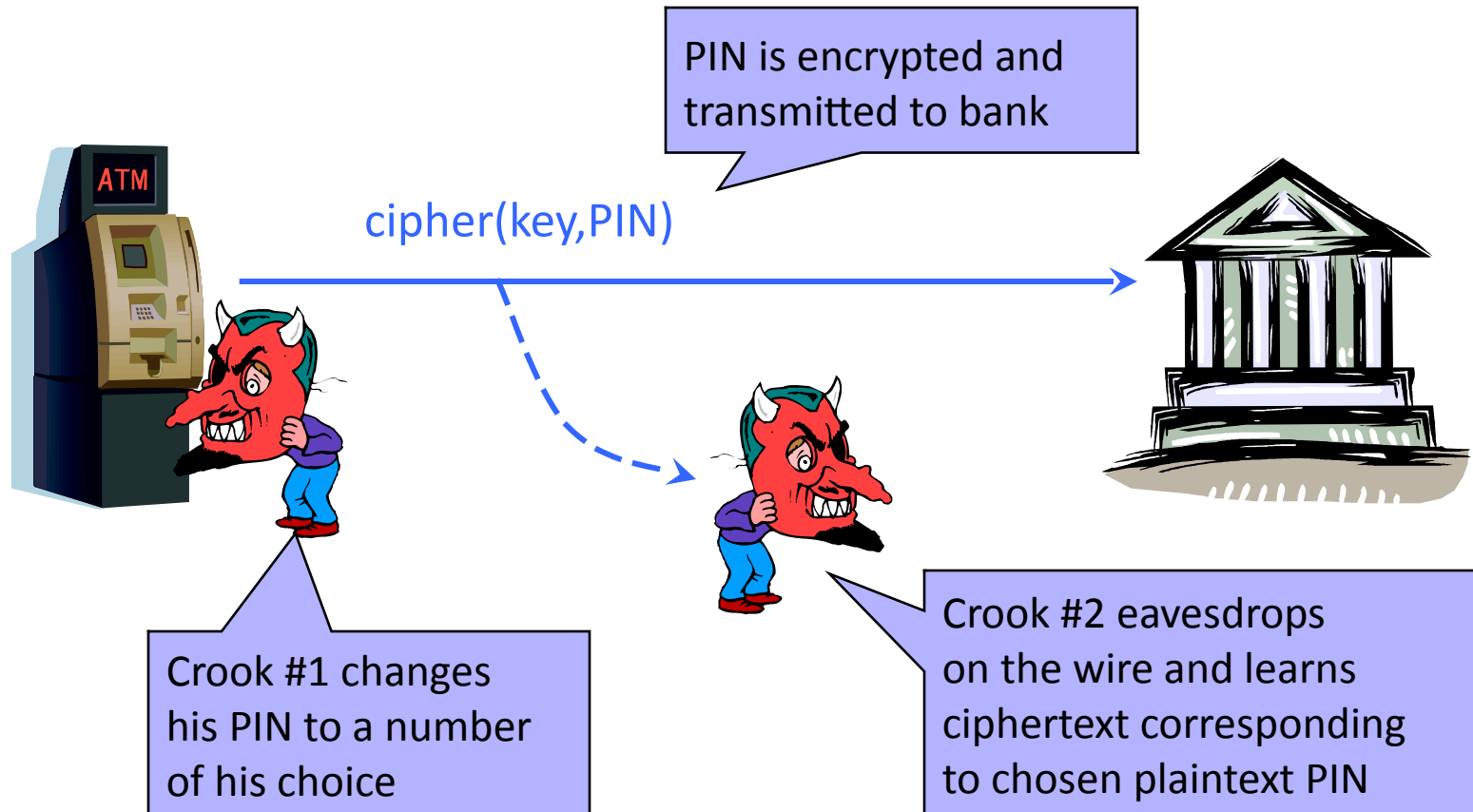
# Known-Plaintext Attack

[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



... repeat for any PIN value



# Diagram Transcription

---

Crook 1 changes his PIN at an ATM

Pin is encrypted ( $\text{Enc}(\text{PIN}, \text{Key})$ ) and sent over the network

Crook 2 eavesdrops and thus learns a valid encryption of any given plaintext (the PIN entered by Crook 1)

# 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  $M_b$ 
  - 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  $\text{Enc}(-, -, b)$

Let  $X, Y$  be any two different plaintexts

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

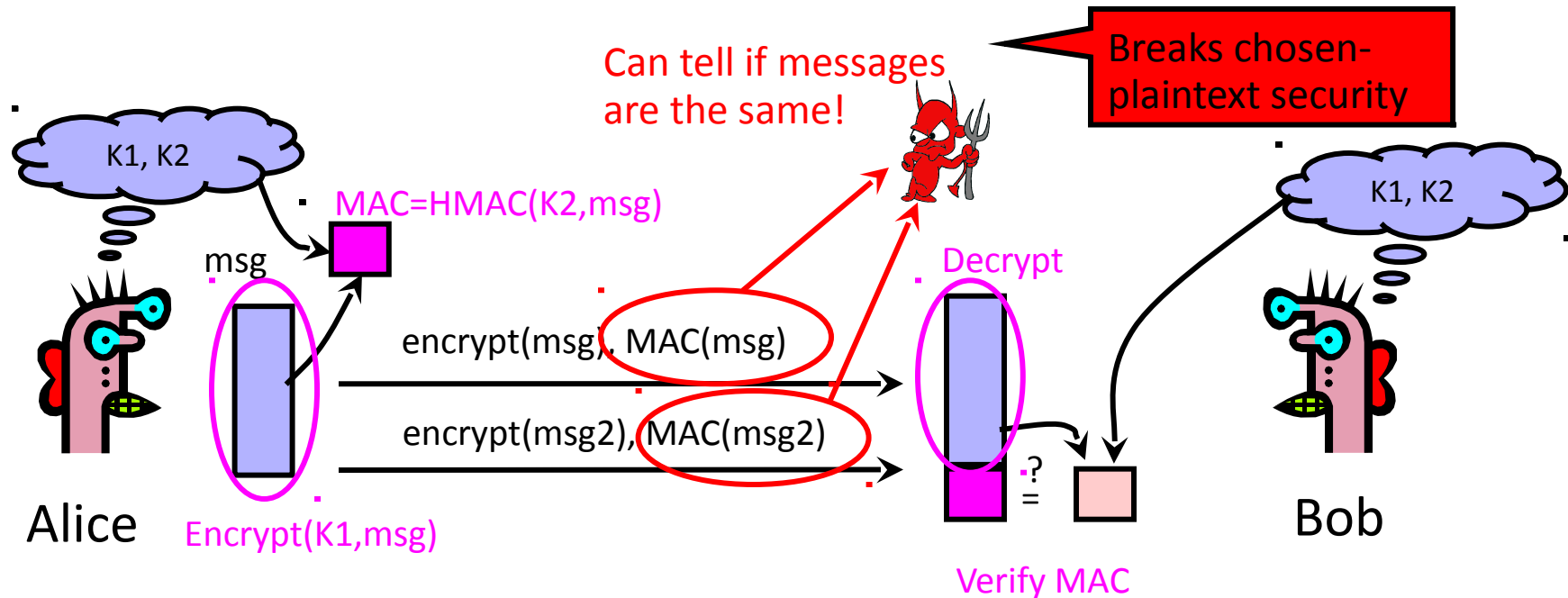
If  $C_1 = C_2$  then  $b=0$  else  $b=1$

◆ The advantage of this attacker A is 1

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

# Encrypt + MAC

Goal: confidentiality + integrity + authentication



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

**Solution:** Encrypt, then MAC (what about MAC, then encrypt?)

# Diagram Transcription

---

If the MAC of the *plaintext* is appended to the *ciphertext* before being sent out, then an attacker can know if two messages contain the same plaintext (because the MACs will match). This breaks chosen plaintext security.