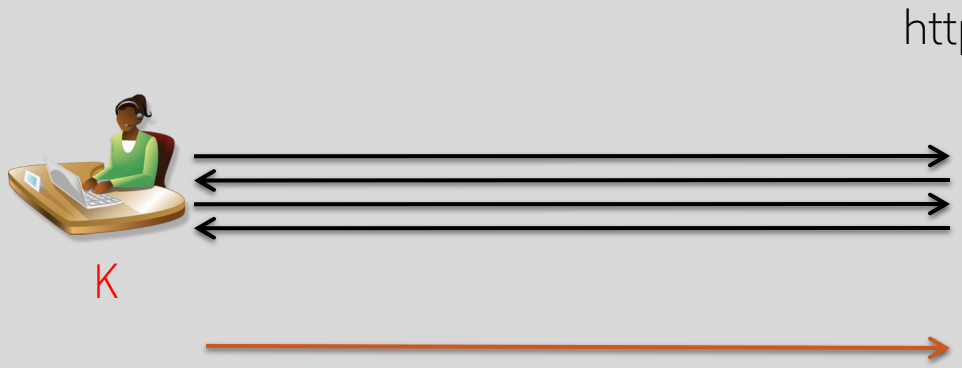
BASICS OF SYMMETRIC CRYPTOGRAPHY

VITALY SHMATIKOV

Online Shopping with HTTPS (HTTP over TLS)



Step 1:
Key exchange protocol
to share secret K

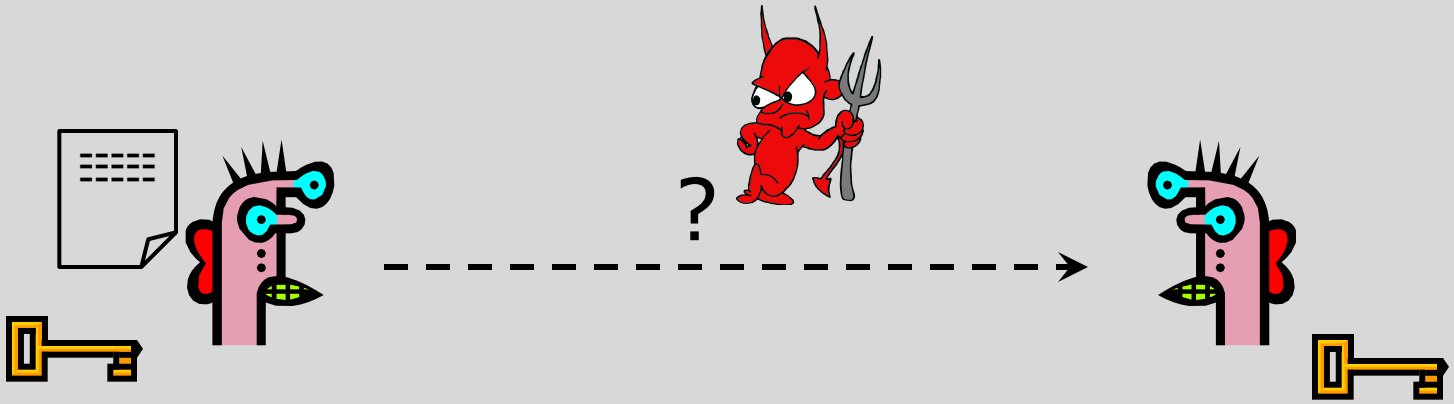
Step 2:
Send data via
secure channel

TLS uses many cryptographic primitives:

key exchange: hash functions, digital signatures, public-key encryption
secure channel: symmetric encryption, message authentication + integrity

Mechanisms to resist replay attacks, man-in-the-middle attacks, truncation attacks, etc…

Goal: Confidentiality



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

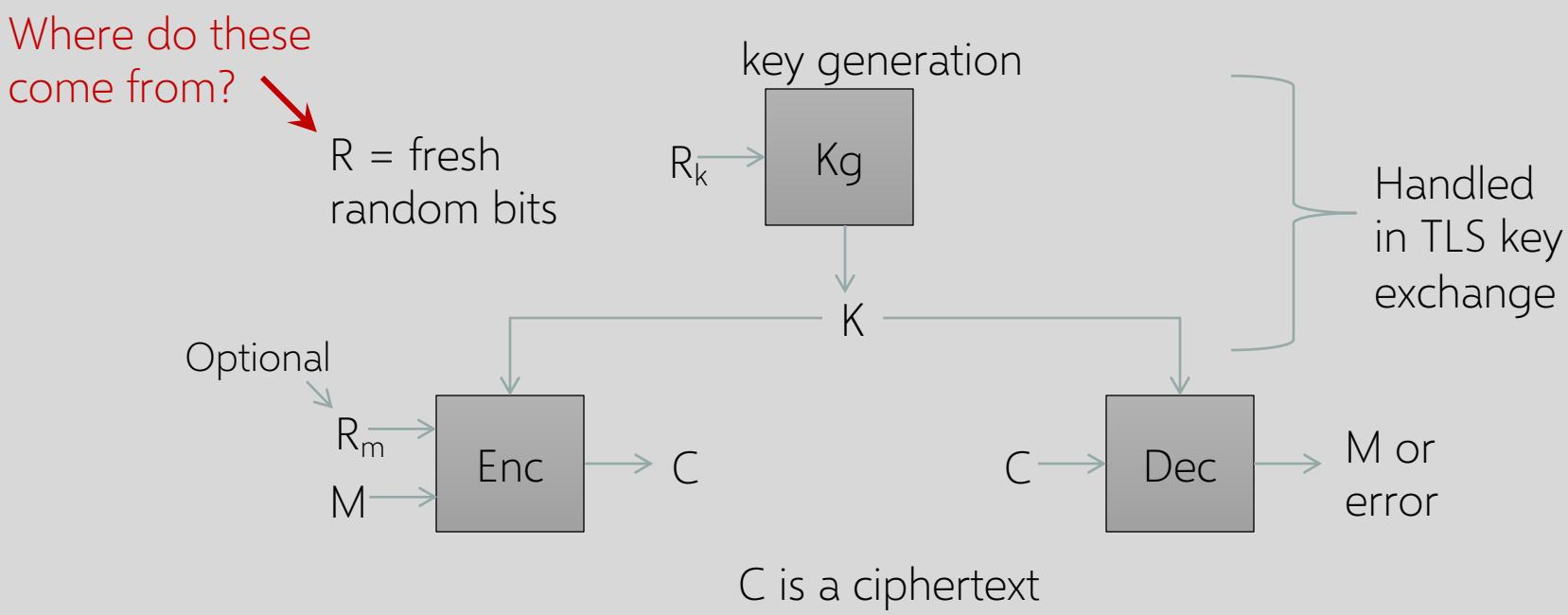
- Attacker knows all algorithms
- Attacker does not know random numbers

Do not rely on secrecy of the algorithms ("security by obscurity")

Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof

Easy lesson:
use a good random number generator!

Symmetric encryption



Correctness: Dec(K, Enc(K,R,M)) = M with probability 1 over randomness R used

Kerckhoffs' principle: which parts are public and which are secret?

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 M

By JOHN MARKOFF
Published: February 14, 2012

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

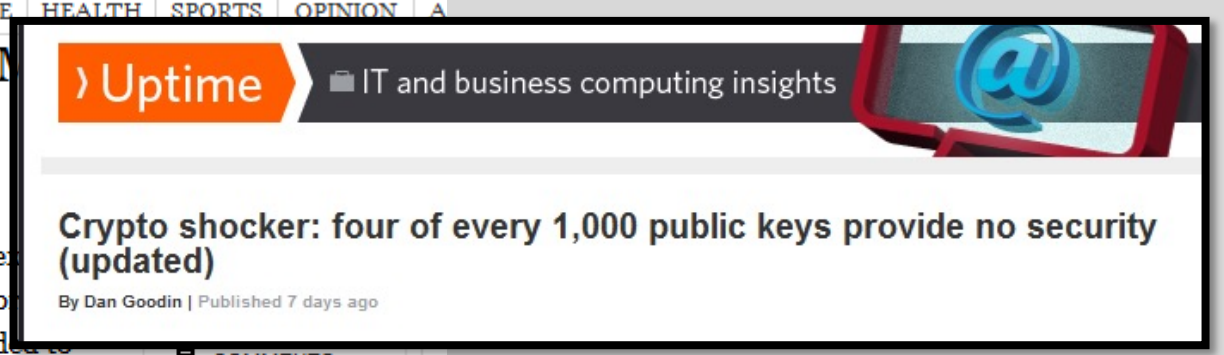
The flaw — which involves a small but measurable number of cases — has to do with the way the system generates random numbers, which are used to make it practically impossible for an attacker to unscramble digital messages. While it can affect the transactions of individual Internet users, there is nothing an individual can do about it. The operators of large Web sites will need to make changes to ensure the security of their systems, the researchers said

Readers' Comments

Readers shared their thoughts on this article.

Read All Comments (127) »

COMMENTS (127)
E-MAIL
PRINT
SINGLE PAGE
REPRINTS
SHARE



› 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

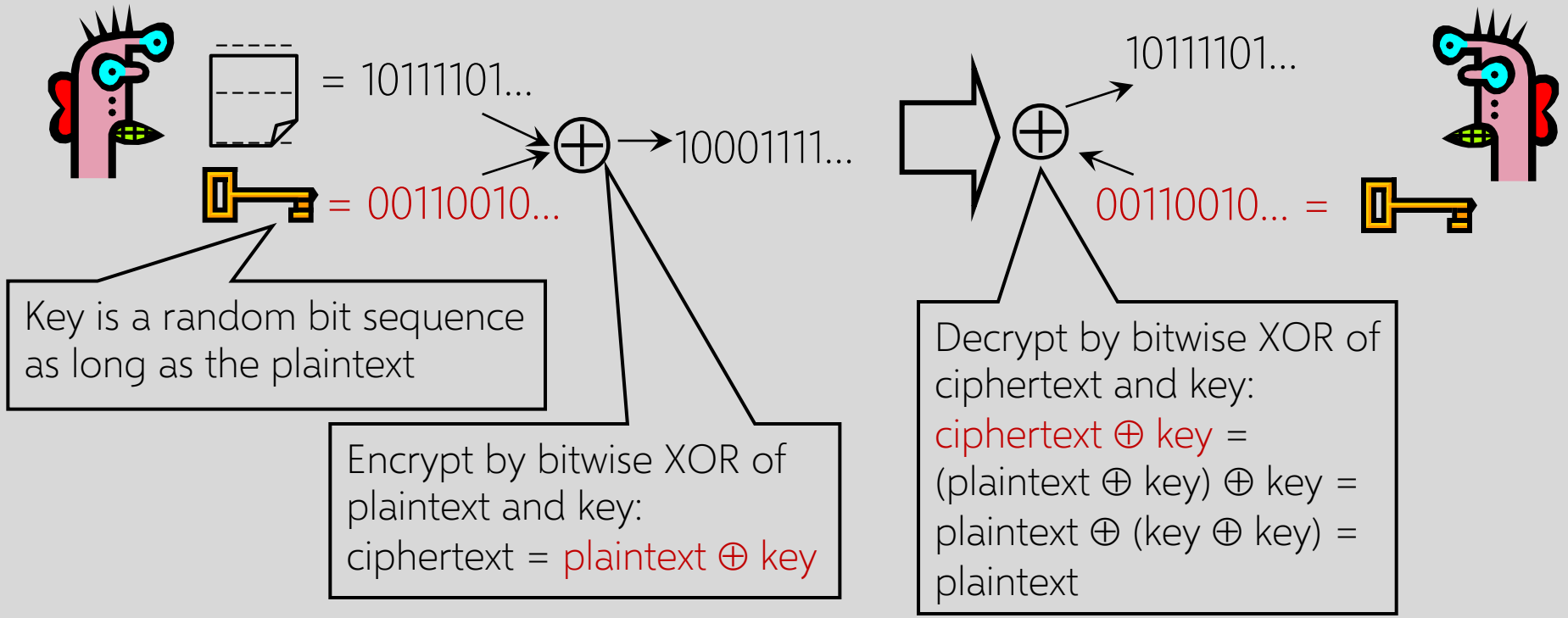
WW2 German Enigma machine

- Polyalphabetic substitution cipher
  - Substitution table changes from character to character
  - Rotors control substitutions

Allies broke Enigma (even before the war), significant intelligence impact

Computers were built to break WW2 ciphers, by Alan Turing and others

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,

<u>if and only if</u> the key sequence is truly random

- True randomness is expensive to obtain in large quantities

<u>if and only if</u> each key is as long as the plaintext

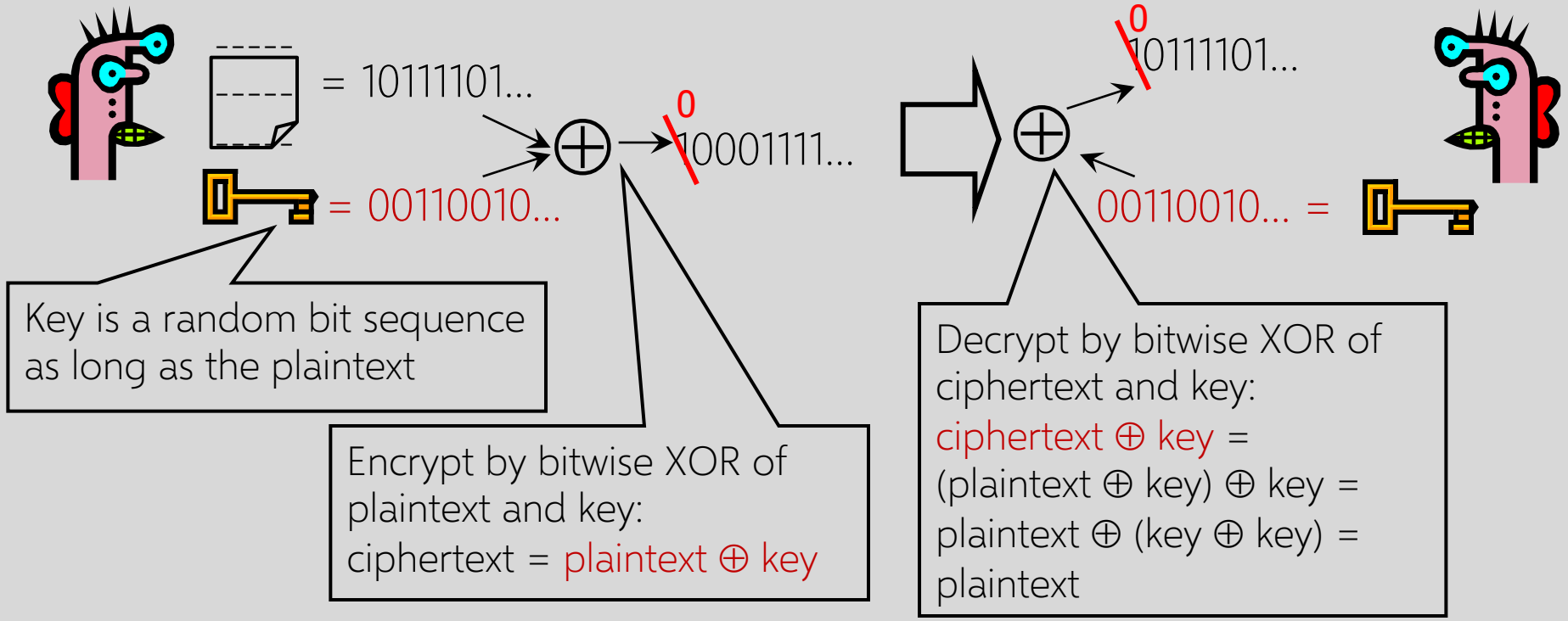
- But how do the sender and the receiver communicate the key to each other? Where do they store the key?

Problems with One-Time Pad

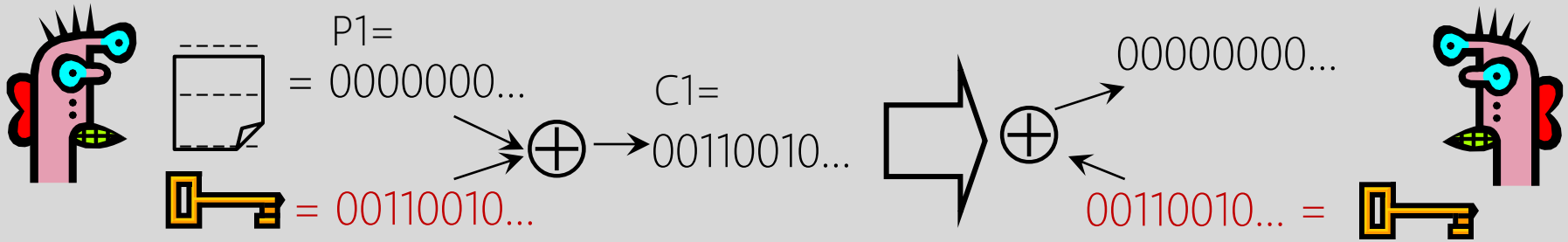
- Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- Does not guarantee integrity
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plaintext, but can easily change it to something else
-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

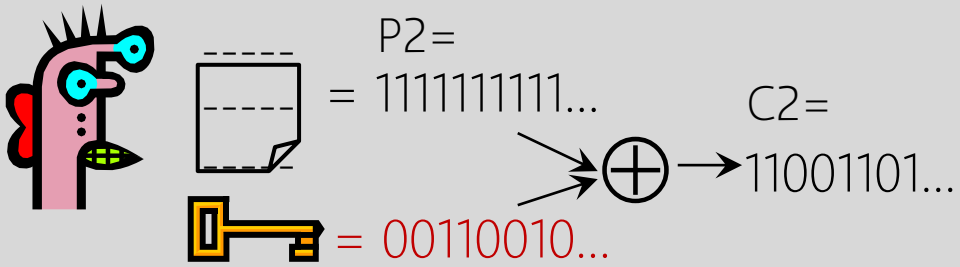
No integrity

No Integrity



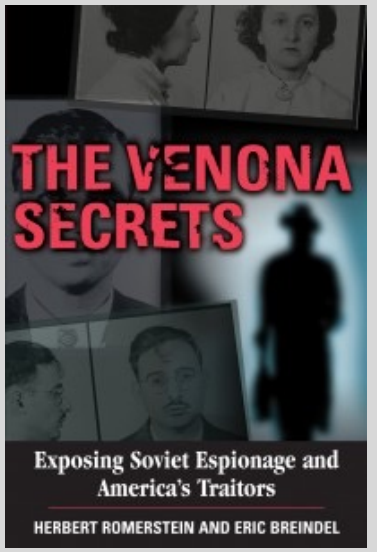
Dangers of Key Reuse





Eavesdropper learns a relationship between plaintexts

C1⊕C2 = (P1⊕K)⊕(P2⊕K) = (P1⊕P2)⊕(K⊕K) = P1⊕P2



Need Small, Reusable Keys

- Use special cryptographic primitives: stream ciphers, block ciphers
- Single key can be re-used (with some restrictions)
- Not as theoretically secure as one-time pad

Stream Ciphers

One-time pad: Ciphertext(Key,Message)=Message⊕Key

- Key must be a random bit sequence as long as the message

Idea: replace "random" with "pseudo-random"

- Use a pseudo-random number generator (PRNG)
- PRNG takes a short, truly random secret seed and expands it into a long "random-looking" sequence
  - E.g., 128-bit seed into a 10^6-bit pseudo-random sequence

No efficient algorithm can tell this sequence from truly random

Ciphertext(Key,Msg)=IV, Msg⊕PRNG(IV,Key)

- Message processed bit by bit (unlike block cipher)

Stream Cipher Terminology

The seed of a pseudo-random generator typically consists of initialization vector (IV) and key

- The key is a secret known only to the sender and the recipient, not sent with the ciphertext
- IV is usually sent with the ciphertext

The pseudo-random bit stream produced by PRNG(IV,key) is referred to as the keystream

Encrypt message by XORing with keystream:

$$\text{ciphertext} = \text{message} \oplus \text{keystream}$$

Properties of Stream Ciphers

- Usually very fast (faster than block ciphers)
  - Used where speed is important: WiFi, DVD, RFID, VoIP
- Unlike one-time pad, stream ciphers do not provide perfect secrecy
  - Only as secure as the underlying PRNG
  - If used properly, can be as secure as block ciphers
- PRNG must be cryptographically secure

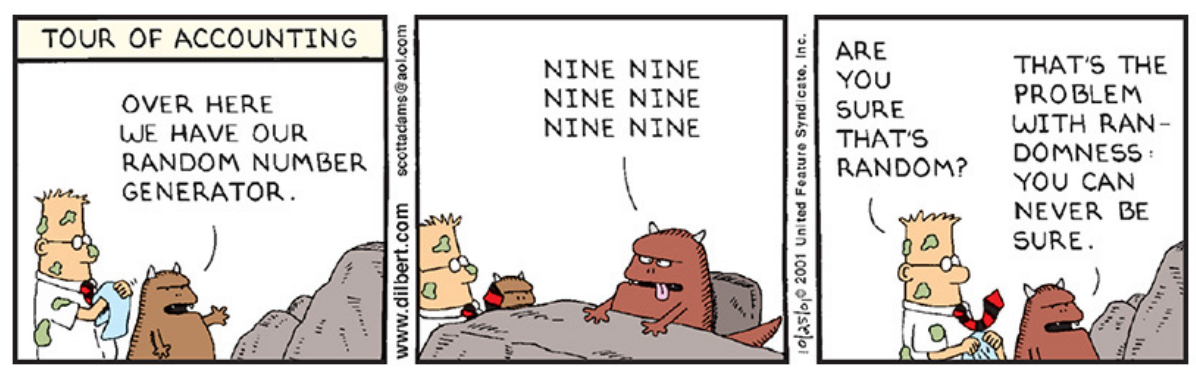
How Random is “Random”?

0XFFFFFFFF EVERY TIME IS 0XDEADBEEF —

How a months-old AMD microcode bug destroyed my weekend [UPDATED]

AMD shipped Ryzen 3000 with a serious microcode bug in its random number generator.

JIM SALTER - 10/29/2019, 7:00 AM



Cryptographically Secure PRNG

Common PRNGs are not cryptographically secure

- Example: LFSR generator in DVD Content Scrambling System, broken in the early 2000s

RC4

- Popular stream cipher from Ron Rivest, many weaknesses, not recommended anymore

Many modern alternatives

- For example, CTR modes of block ciphers such as AES

Jon Lech Johansen ("DVD Jon")

Using Stream Ciphers

No integrity

- Associativity & commutativity:

  $(M_1 \oplus PRNG(seed)) \oplus M_2 = (M_1 \oplus M_2) \oplus PRNG(seed)$

- Need an additional integrity protection mechanism

Known-plaintext attack is very dangerous if keystream is ever repeated !

- Self-cancellation property of XOR: $X \oplus X = 0$
- $(M_1 \oplus PRNG(seed)) \oplus (M_2 \oplus PRNG(seed)) = M_1 \oplus M_2$
- If attacker knows M_1, then easily recovers M_2 … also, most plaintexts contain enough redundancy that can recover parts of both messages from $M_1 \oplus M_2$

Case Study: WEP (Wired Equivalent Privacy)

Original Wi-Fi encryption in the 802.11b standard

Goals: confidentiality, integrity, authentication

- Intended to make wireless as secure as wired network

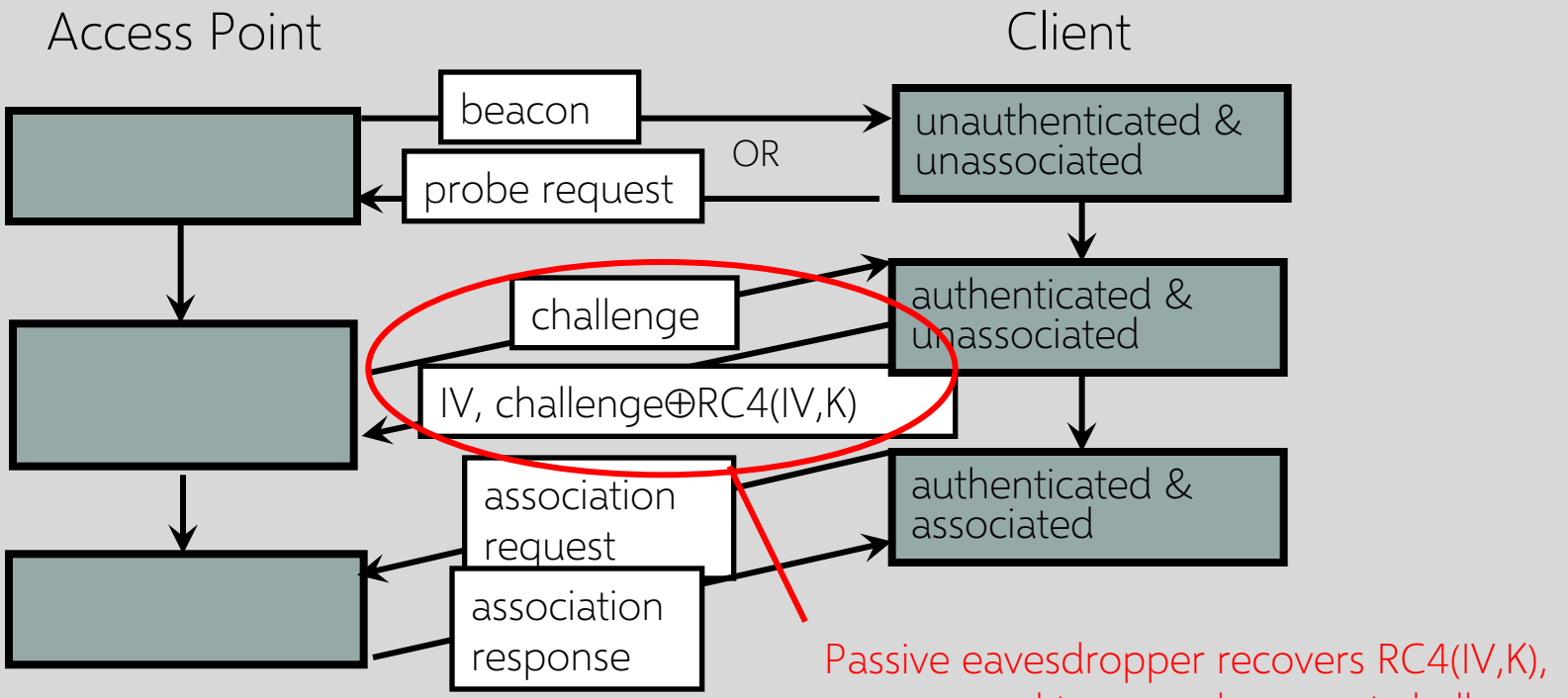
Assumes that a secret key is shared between access point and client

Uses RC4 stream cipher seeded with 24-bit initialization vector and 40-bit key

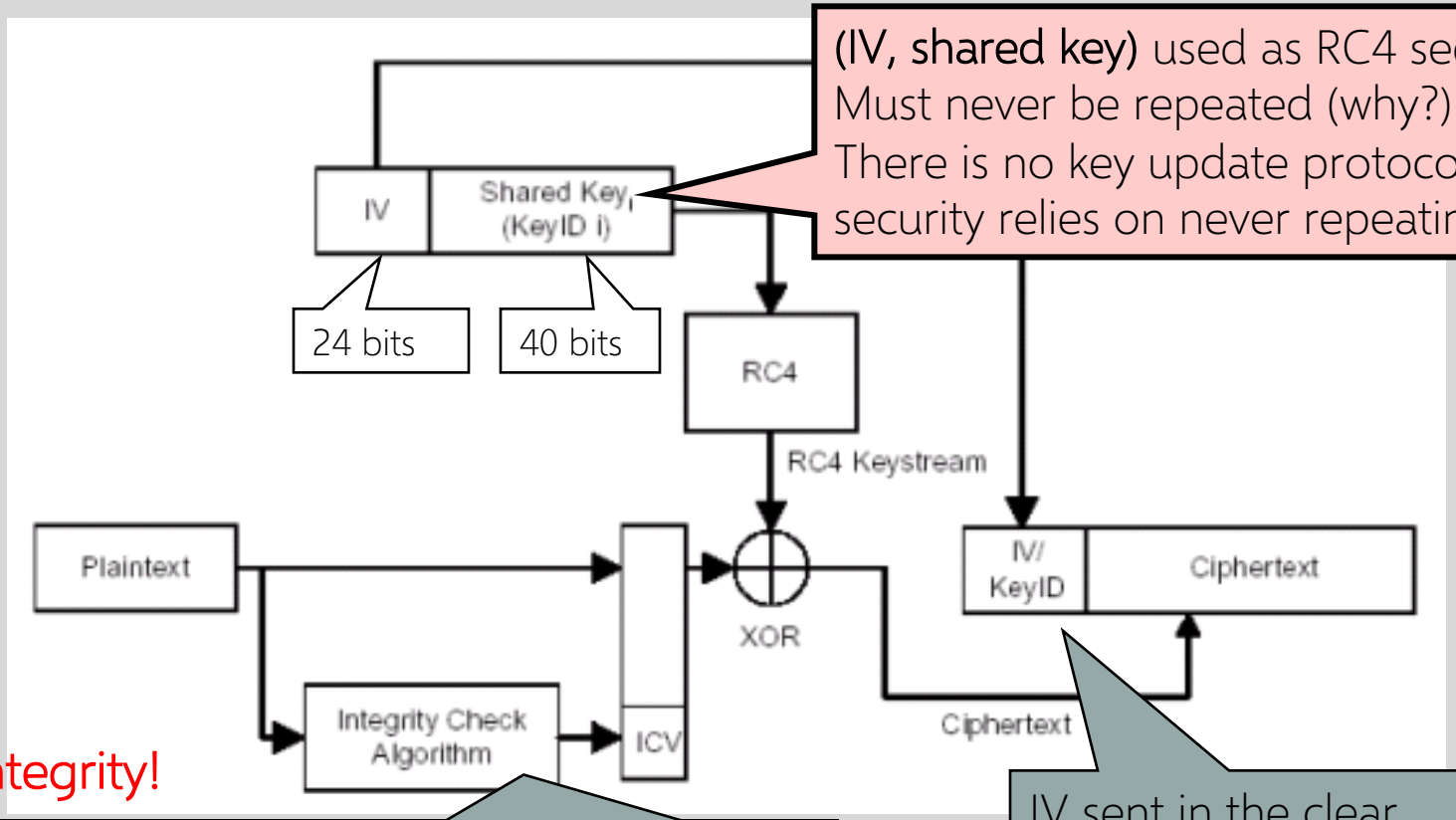
Terrible design choice for wireless environment

Shared Key Authentication

Prior to communicating data, access point may require client to authenticate



How WEP Works



(IV, shared key) used as RC4 seed
Must never be repeated (why?)
There is no key update protocol, so security relies on never repeating IV

no integrity!

CRC-32 checksum is linear in ⊕:
if attacker flips some plaintext bits, he knows which bits of CRC to flip to produce the same checksum

IV sent in the clear
Worse: **changing IV with each packet is optional!**

RC4 Was a Bad Choice for Wireless

Stream ciphers require sender and receiver to be at the same place in the keystream

- Not suitable when packet losses are common

WEP solution: a separate keystream for each packet (requires a separate seed for each packet)

- Can decrypt a packet even if a previous packet was lost

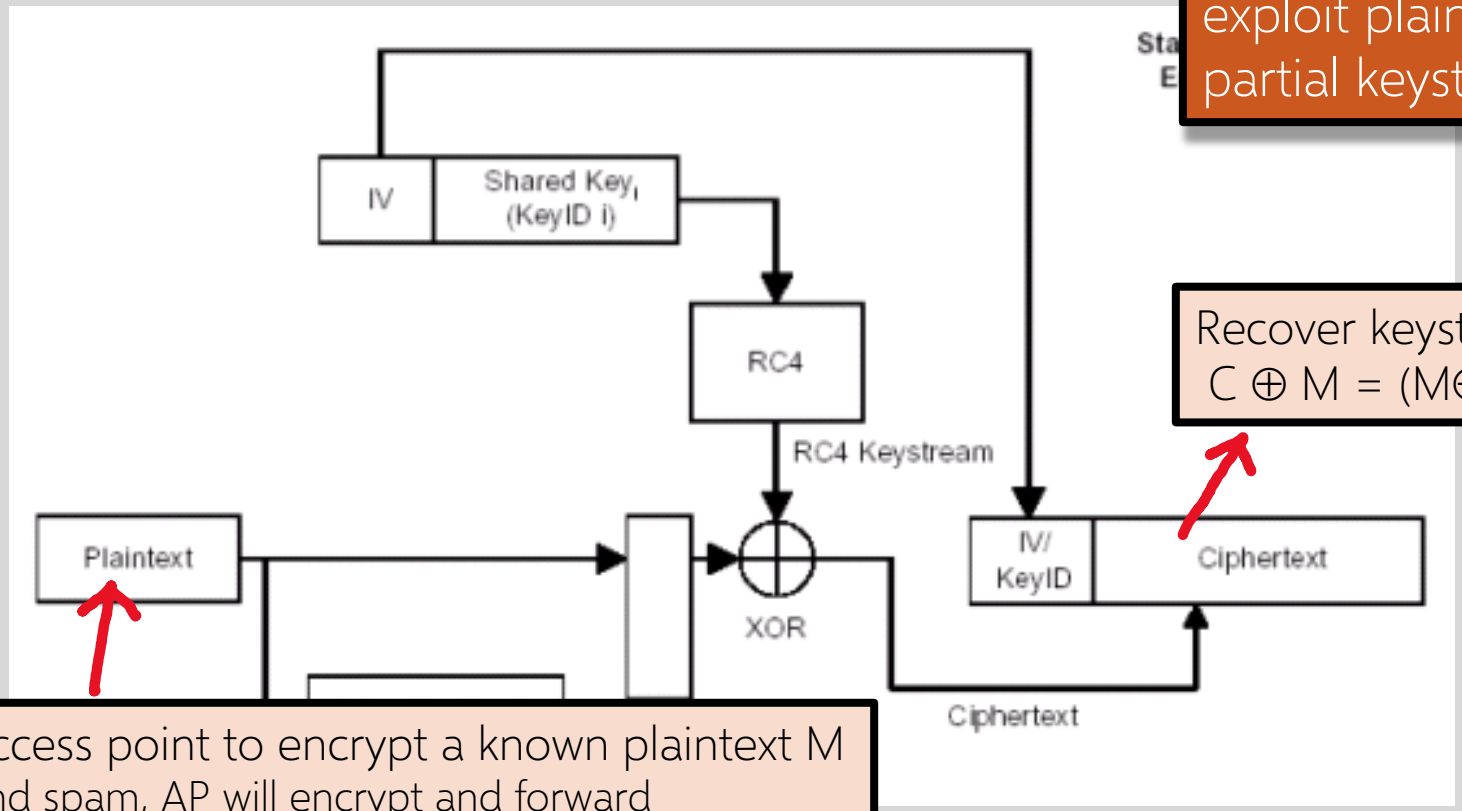
But there aren't enough possible seeds!

- RC4 seed = 24-bit initialization vector + fixed key
- Assuming 1500-byte packets at 11 Mbps,
  2^{24} possible IVs will be exhausted in about 5 hours

Seed reuse is deadly for stream ciphers

Recovering the Keystream

Plaintexts are not random (for example, IP packet structure is very regular).
Even without knowing the plaintext, can exploit plaintext regularities to recover partial keystream.



Recover keystream:
$C \oplus M = (M \oplus RC4(IV,key)) \oplus M = RC4(IV,key)$

Get access point to encrypt a known plaintext M
- Send spam, AP will encrypt and forward
- Get victim to send an email with known content

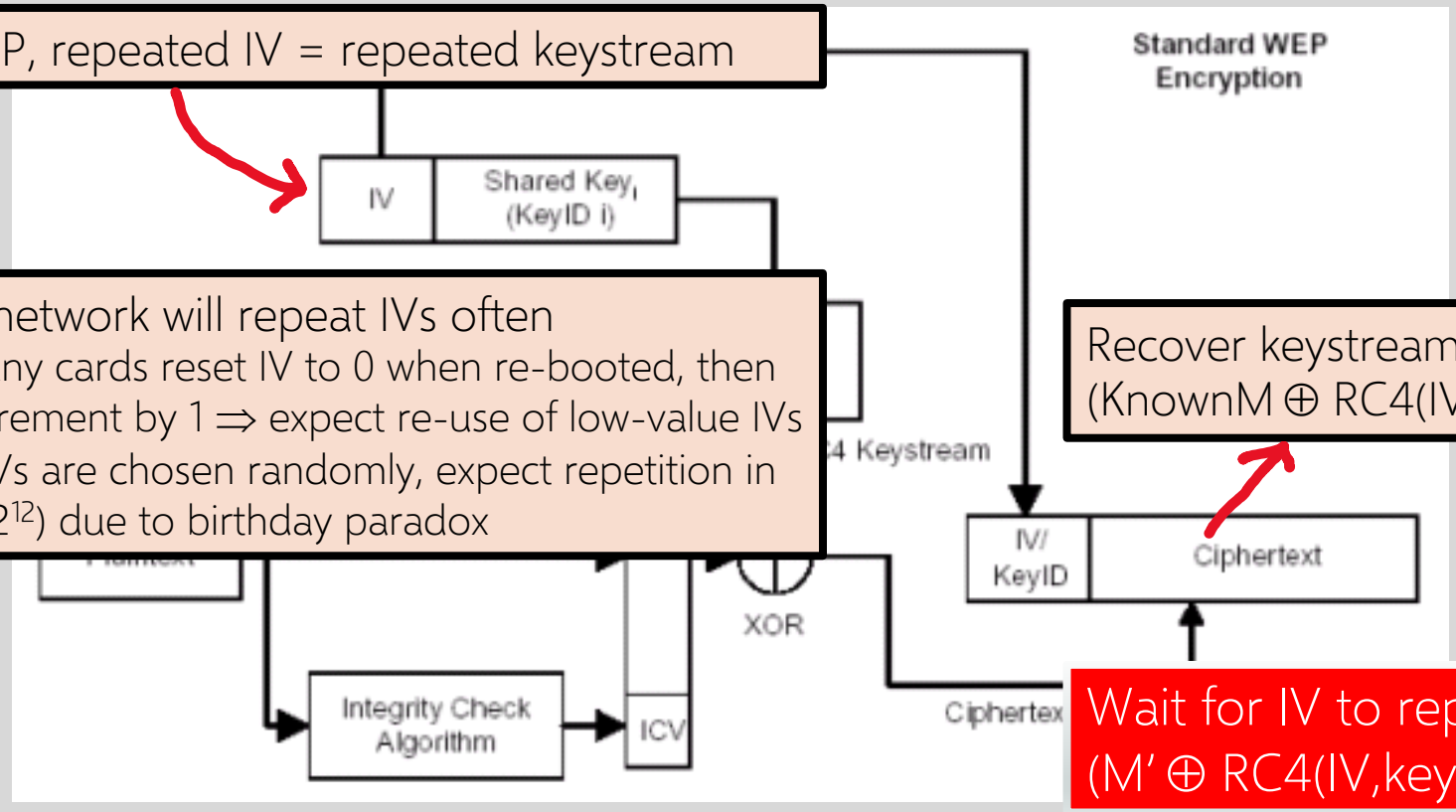
Not a problem if the keystream is never re-used!

Keystream Will Be Re-Used

In WEP, repeated IV = repeated keystream

Busy network will repeat IVs often

- Many cards reset IV to 0 when re-booted, then increment by 1 ⇒ expect re-use of low-value IVs
- If IVs are chosen randomly, expect repetition in $O(2^{12})$ due to birthday paradox



Recover keystream for each IV, store in table:
(KnownM ⊕ RC4(IV,key)) ⊕ KnownM = RC4(IV,key)

Wait for IV to repeat, decrypt, enjoy plaintext:
(M' ⊕ RC4(IV,key)) ⊕ RC4(IV,key) = M'

It Gets Worse

- Misuse of RC4 in WEP is a design flaw with no fix
  - Longer keys do not help! The problem is re-use of IVs, their size is fixed (24 bits)
  - Attacks are passive and very difficult to detect
- Perfect target for the Fluhrer et al. attack on RC4
  - Attack requires known IVs of a special form
  - WEP sends IVs in plaintext
  - Generating IVs as counters or random numbers will produce enough "special" IVs in a matter of hours
- This results in key recovery (not just keystream)
  - Can decrypt even ciphertexts whose IV is unique

Fix in 802.11i

Patch (TKIP): still RC4, but encrypts IVs and establishes new shared keys for every 10 KBytes transmitted

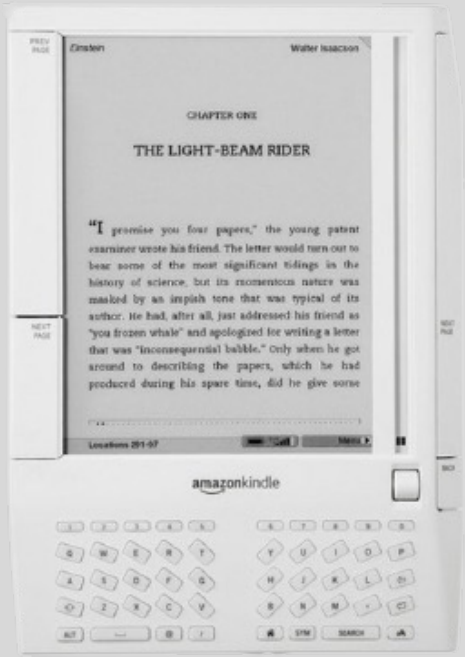
- Use same network card, only upgrade firmware
- Deprecated by the Wi-Fi alliance

Long-term: AES in CCMP mode, 128-bit keys, 48-bit IVs

- Block cipher in a stream cipher-like mode

Many Stream Cipher Attacks Over the Years

MIFARE cards



Kindle DRM

A5/1 GSM cipher

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

A Bit of Block Cipher History

Playfair and variants (from 1854 until WW2)

Feistel structure

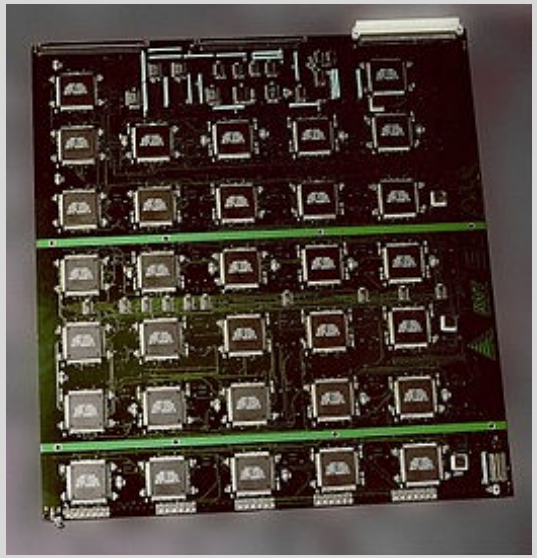
- "Ladder" structure: split input in half, put one half through the round and XOR with the other half… after 3 random rounds, ciphertext indistinguishable from a random permutation

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)

Best Attacks Against DES

Attack	Attack type	Complexity	Year
Biham, Shamir	Chosen plaintexts, recovers key	2^{47} plaintext, ciphertext pairs	1992
Matsui	Known plaintext, ciphertext pairs, recovers key	2^{42} plaintext, ciphertext pairs, ~2^{41} DES computations	1993
DESCHALL	Unknown plaintext, recovers key	$2^{56/4}$ DES computations 41 days	1997
EFF Deep Crack	Unknown plaintext, recovers key	~4.5 days	1998
Deep Crack + DESCHALL	Unknown plaintext, recovers key	22 hours	1999



EFF's custom machine with "Deep Crack" chips

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

Design uses some clever math

Take a cryptography course to learn more!

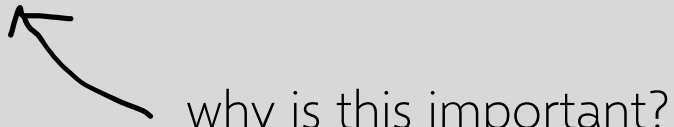
AES in Hardware

x86 instructions implementing AES

- **aesenc, aesenclast:** one round of AES encryption
- **aesdec, aesdeclast:** one round of AES decryption
- **aeskeygenassist**: AES key expansion

10x faster than AES in software

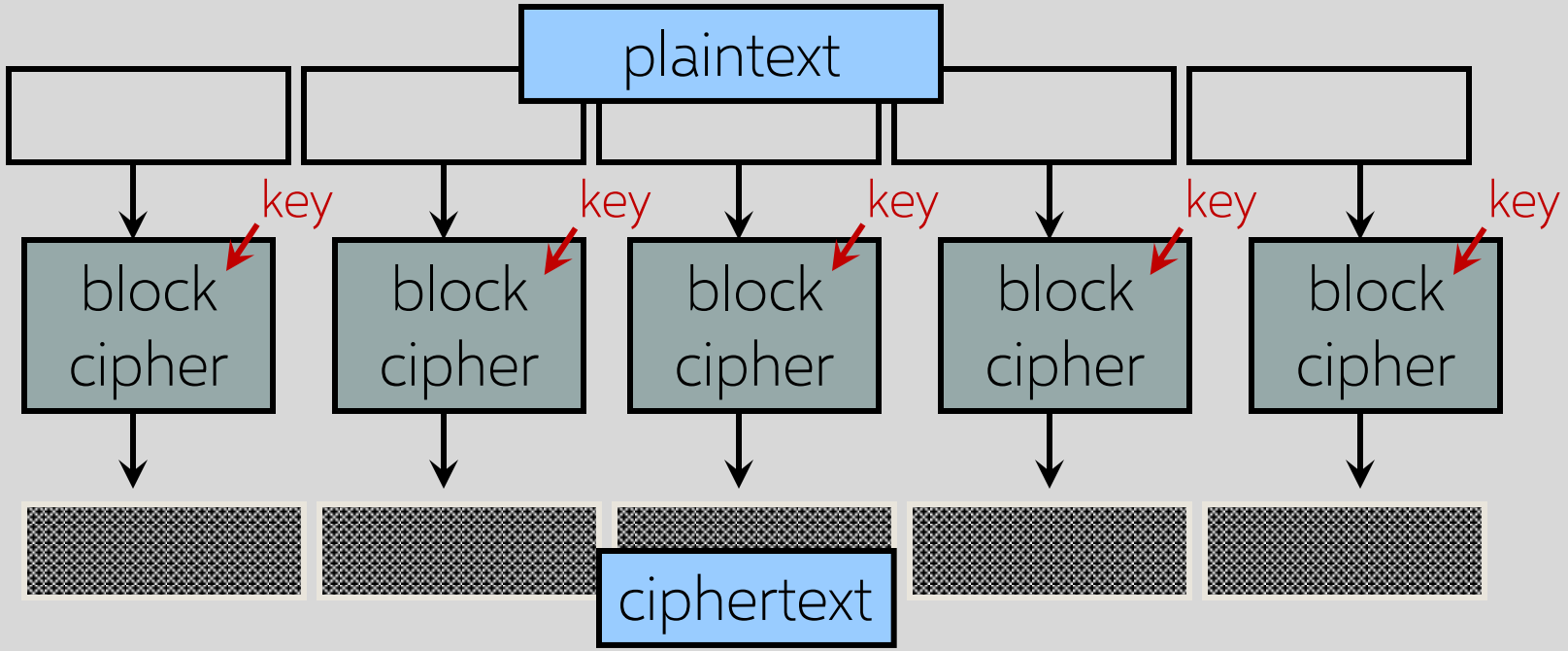
All AES instructions execute in constant time



Encrypting a Large Message

So, we've got a good block cipher, but our message is longer than 128 bits...

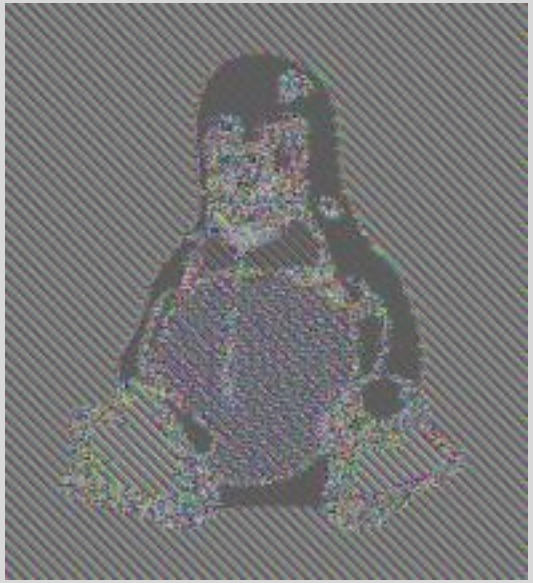
Electronic Code Book (ECB) Mode



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

ECB Mode Leaks Information!

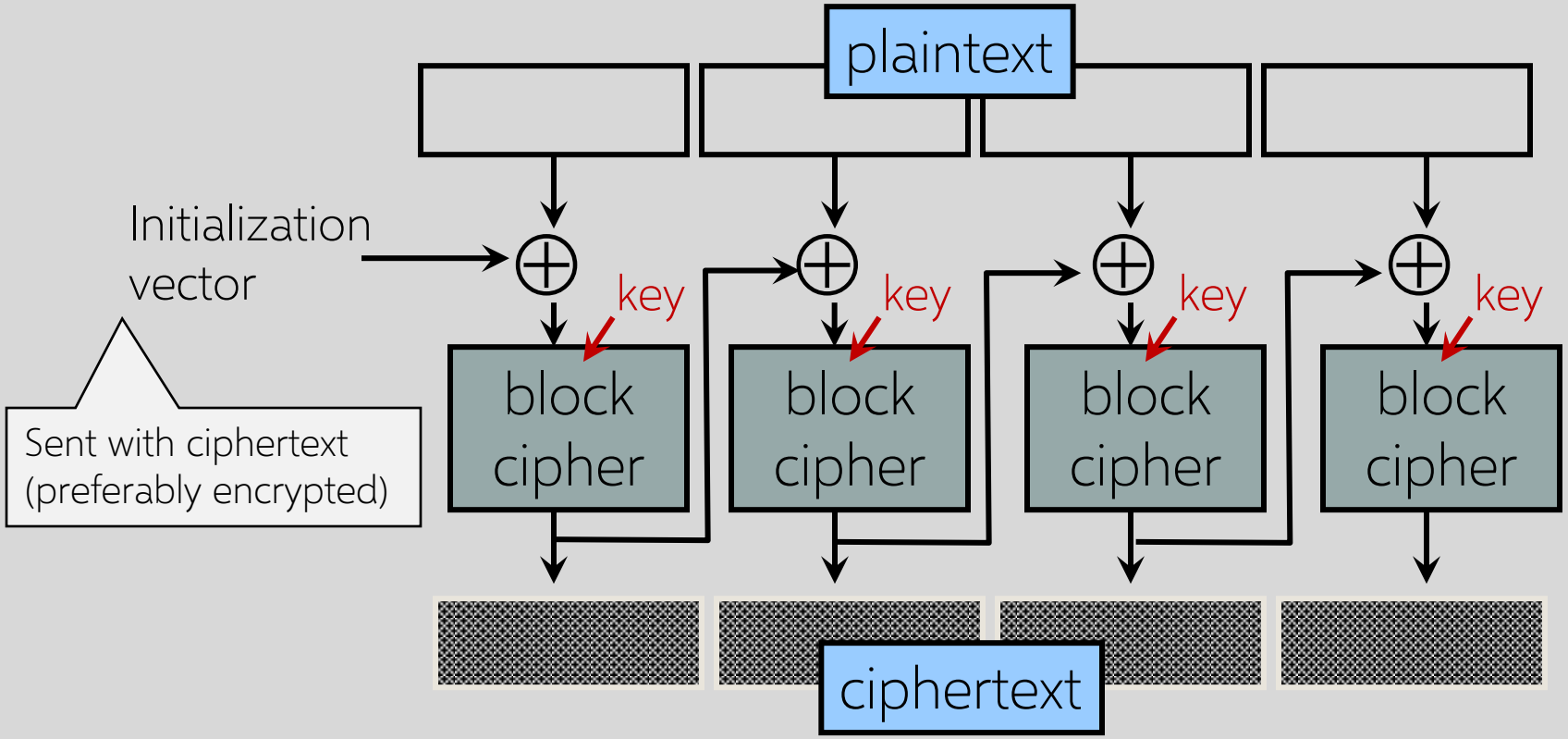
Encrypt in ECB mode



Remember Adobe Passwords Leak?

- 153 million account passwords leaked in 2013
- Encrypted using 3DES in ECB mode rather than hashed

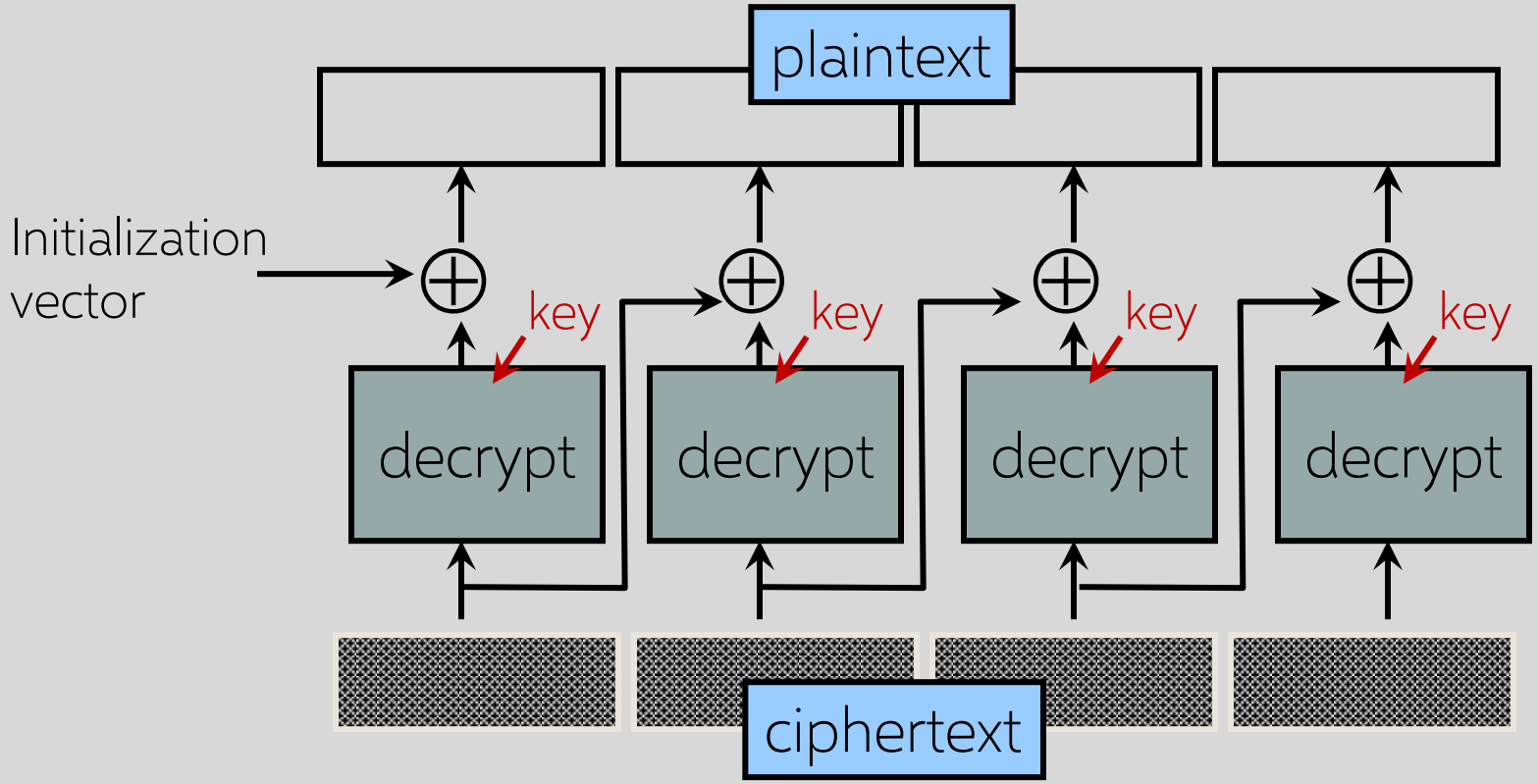
Cipherblock Chaining (CBC) Mode: Encryption



- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext

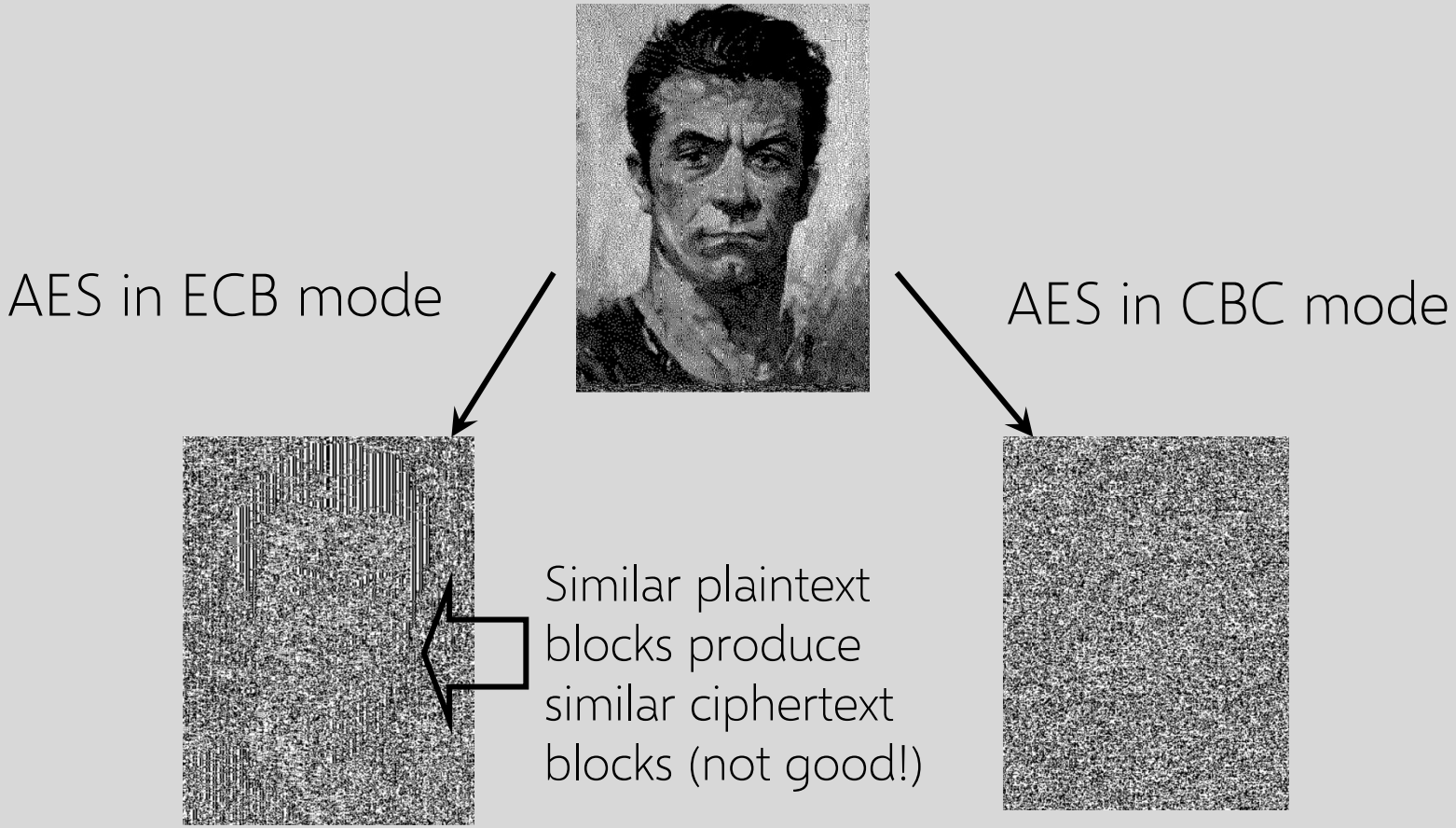
Still does not guarantee integrity...

CBC Mode: Decryption



Picture credit: Bart Preneel

ECB vs. CBC

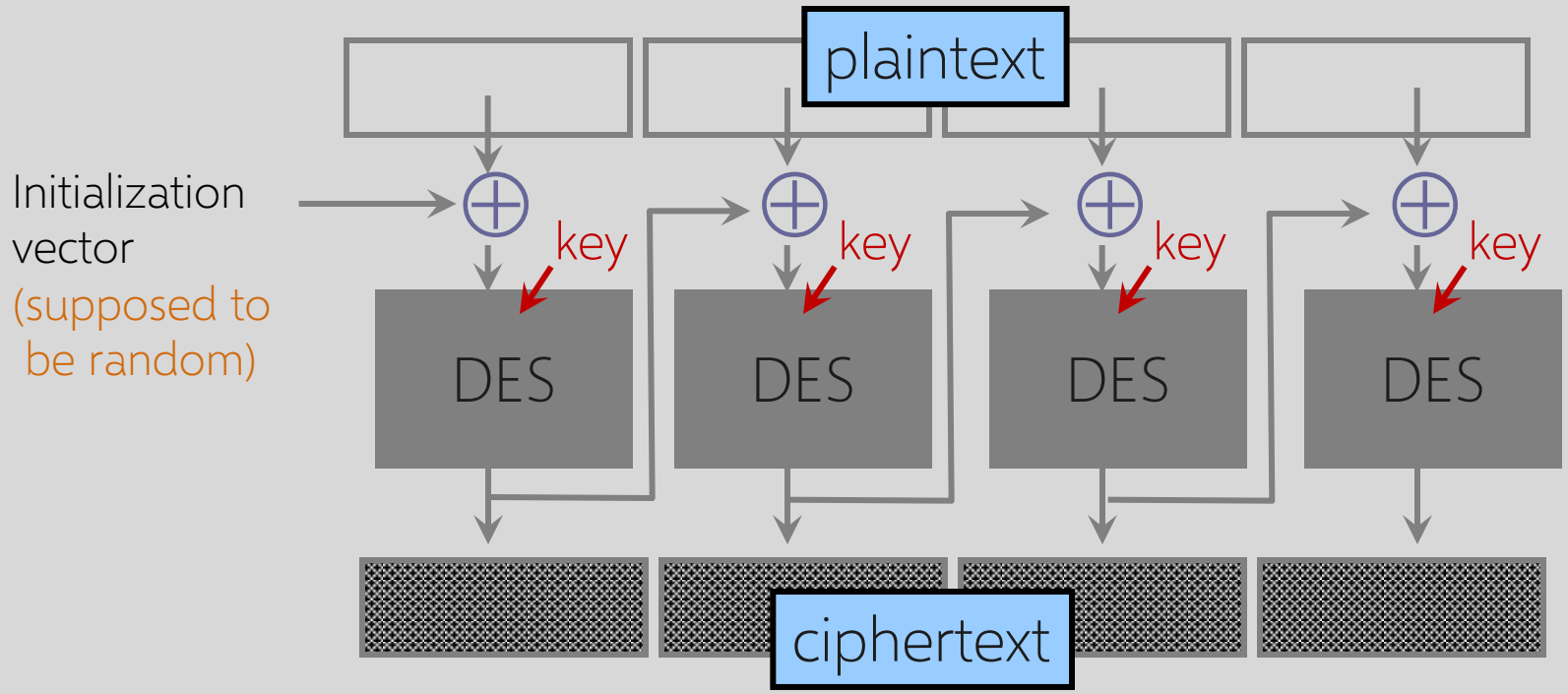


Choosing the Initialization Vector

Key used only once	Key used multiple times	Key used multiple times, unique messages

No IV needed (can use IV=0)	• Best: fresh, random IV for every message
	• Can use unique IV (eg, counter), but then the first step in CBC mode **must** be IV' ← E(k, IV)
	  • Example: Windows BitLocker
	  • May not need to transmit IV with the ciphertext
		• Synthetic IV: IV ← F(k', message)
		  • F is a cryptographically secure keyed pseudorandom function

CBC and Electronic Voting



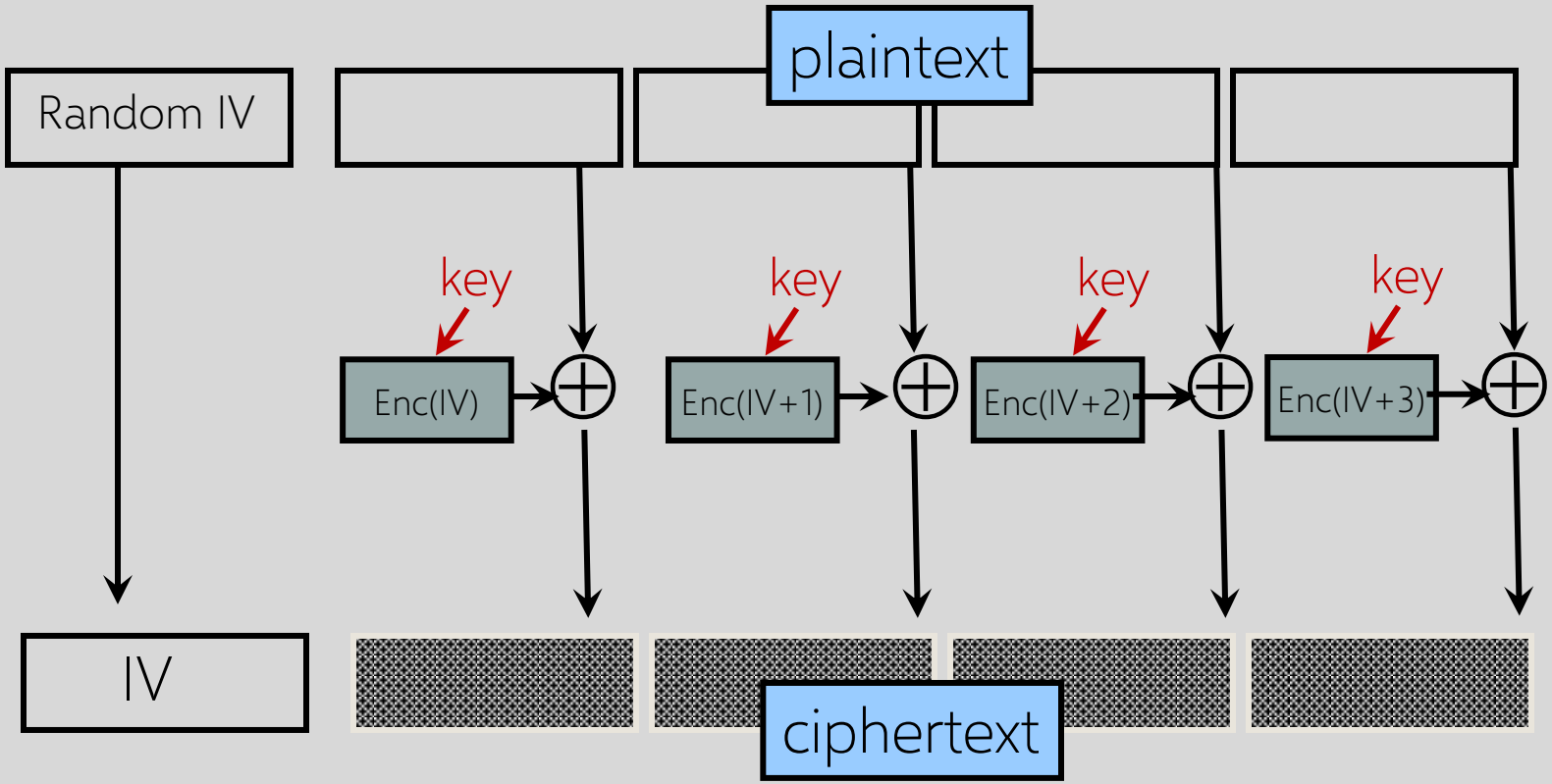
Found in the source code for Diebold voting machines:

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

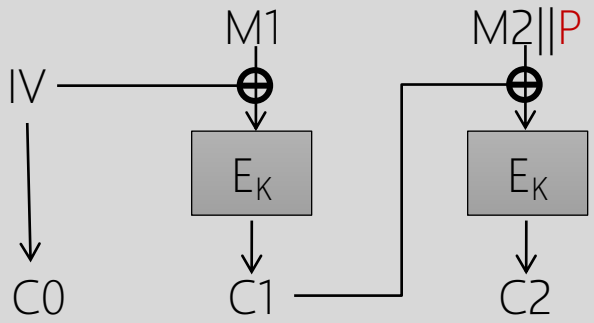
CTR (Counter Mode)

Still does not guarantee integrity

Fragile if counter repeats



Checking If CBC Ciphertext Decrypted Correctly



Assume that
M1||M2 has length 2n-8 bits

P is one byte of padding
that must equal 0x00

Adversary obtains
ciphertext C0,C1,C2

C0, C1, C2

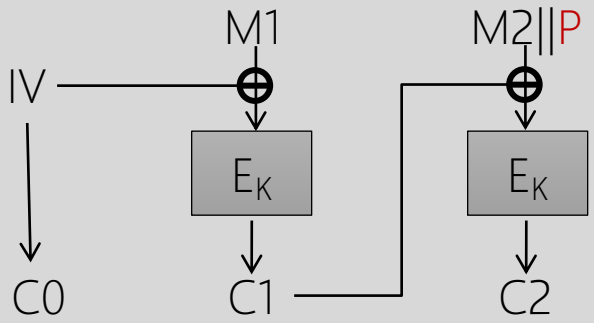
ok

C0, C1 ⊕1, C2

error

Dec(K,C')
M1'||M2'||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
Return error
Else
Return ok

Padding Oracle Attack on CBC Decryption



Assume that
M1||M2 has length 2n-8 bits

P is one byte of padding
that must equal 0x00

Adversary obtains
ciphertext C0,C1,C2

Let R be arbitrary bits

R, C0, C1

error

R, C0⊕1, C1

error

...

R, C0⊕i, C1

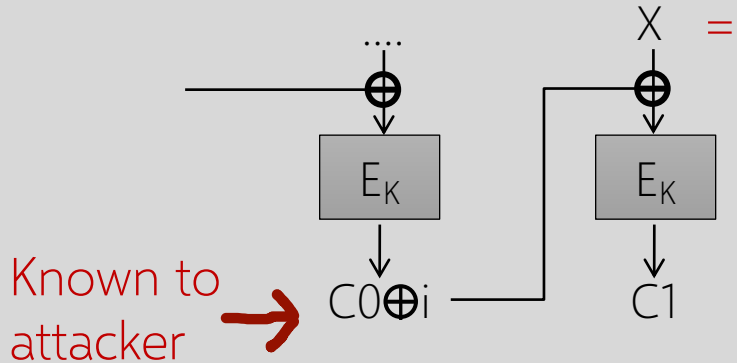
ok!

Dec(K,C')
M1'||M2'||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
Return error
Else
Return ok

High-Level Intuition

Learning last byte of this sum =
learning last byte of unknown plaintext

X = Unknown plaintext ⊕ known value



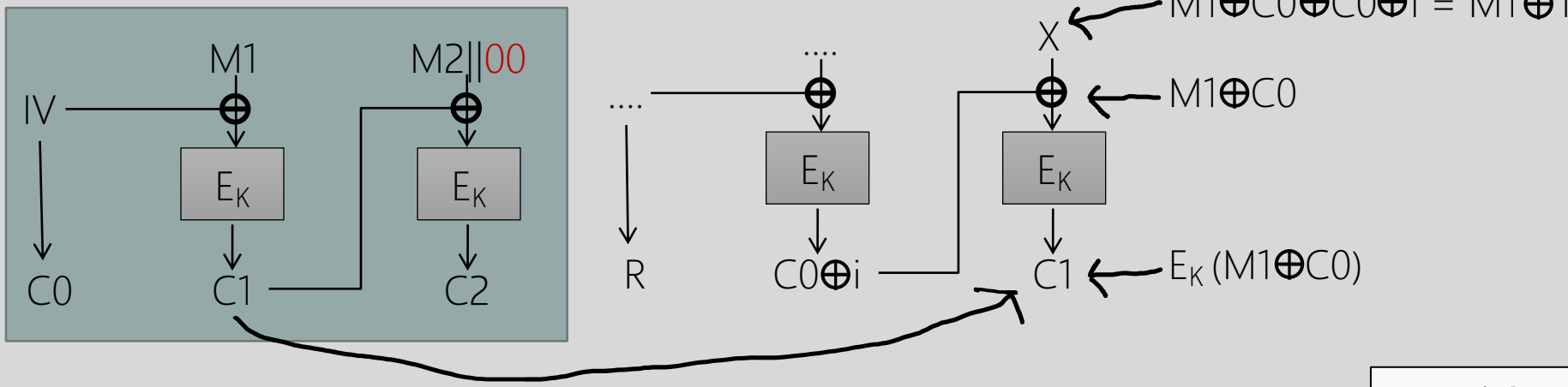
Decryption succeeded, therefore
last byte = known padding value

R, C0 ⊕ i, C1

ok!

Dec(K,C')
M1'||M2'||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
Return error
Else
Return ok

Adversary's Analysis



Last byte of this value must be 00 (why?)

Therefore, last byte of M1 is i

R, C0⊕ i, C1

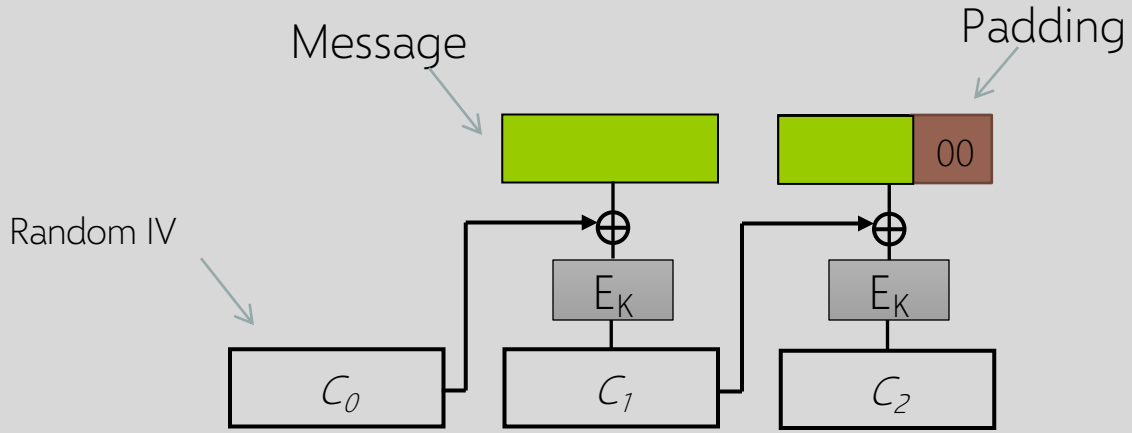
ok!

Adversary obtains ciphertext C0,C1,C2

Let R be arbitrary bits

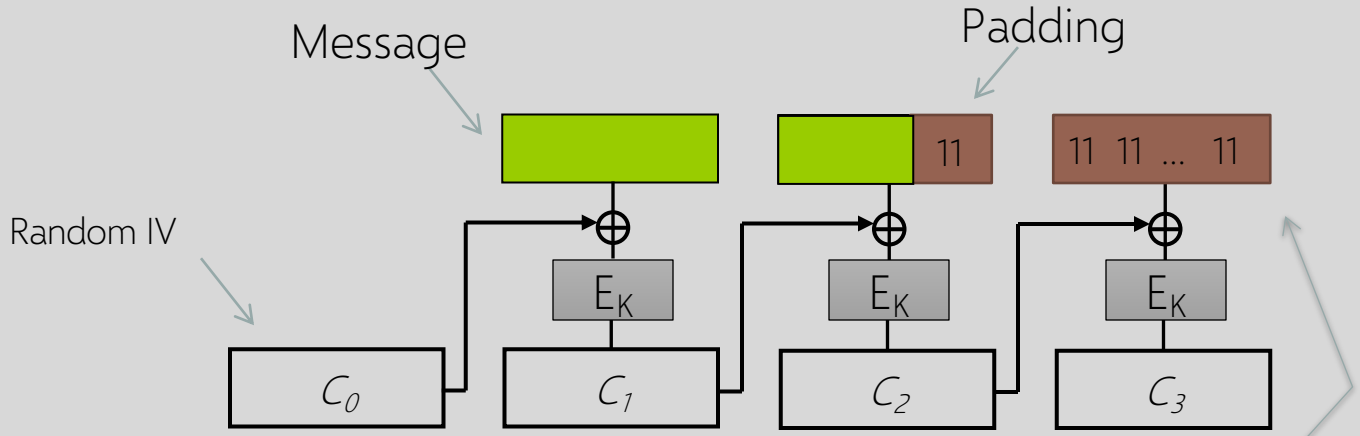
Dec(K,C')
M1'||M2'||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
Return error
Else
Return ok

Padding for CBC Mode in TLS



Possible paddings in TLS: 00, 01 01, 02 02 02, etc.

Padding for CBC Mode in TLS

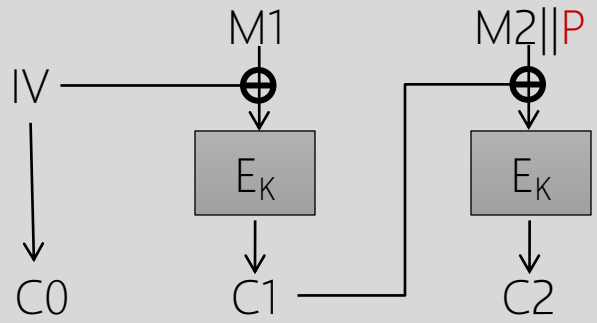


Possible paddings in TLS: 00, 01 01, 02 02 02, etc.

"Lengths longer than necessary might be desirable to frustrate attacks on a protocol that are based on analysis of the lengths of exchanged messages."
RFC 5246

These are called "traffic analysis" attacks

Vaudenay's Padding Oracle Attack



Goal:
decrypt entire plaintext

We know that:
$00 = i \oplus IV[n] \oplus M1[n]$

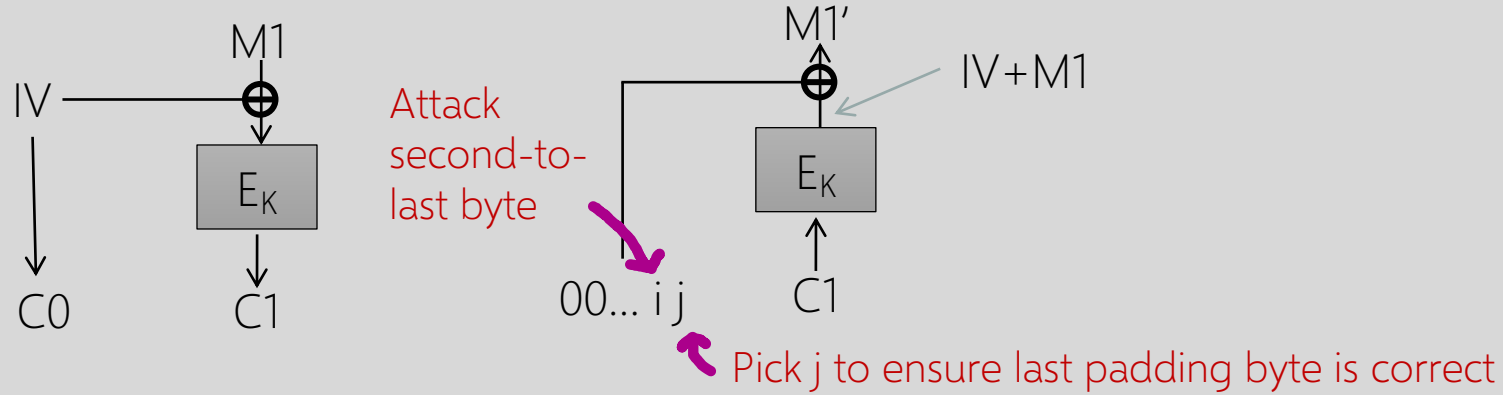
Or do we? Could be:
$01 = i \oplus IV[n] \oplus M1[n]$
$01 = IV[n-1] \oplus M1[n-1]$

Easy to exclude other cases

00...00, C1 → ← error
00...01, C1 → ← error
...
00... i, C1 → ← ok

Dec(K,C')
M1' = CBC-Dec(K,C')
(X,plen) <- lastbyte(M1')
For i = 0 to padlen do
    (X,plen') <- lastbyte(X)
    If plen' != plen return Error
Return Ok

Vaudenay's Padding Oracle Attack



We know M1[n]. Let's get second to last byte.

Solve j to make M1'[n] = 01
01 = j ⊕ IV[n] ⊕ M1[n]

Know that:
01 = i ⊕ IV[n-1] ⊕ M1[n-1]

Repeat for all n bytes

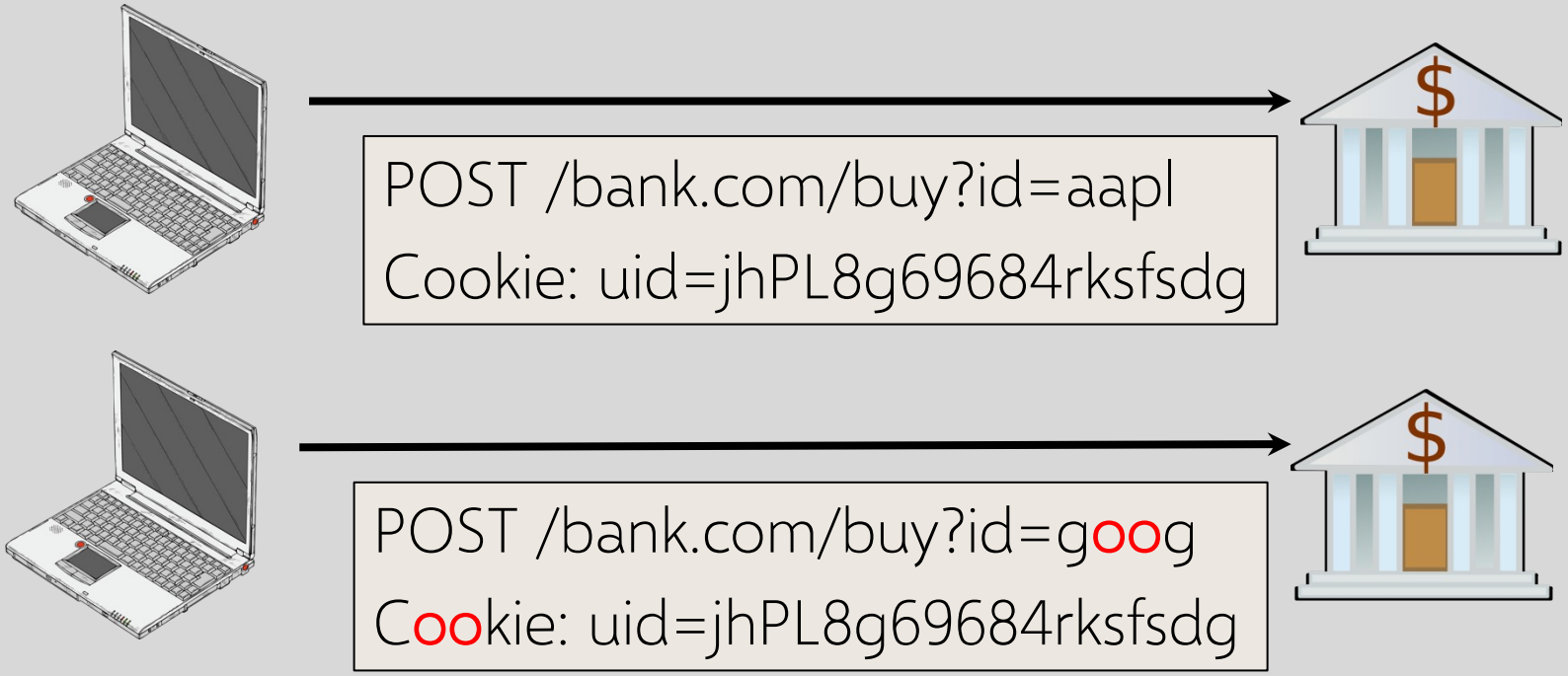
00...00 j, C1
error
00...01 j, C1
error
...
00... i j, C1
ok

Dec(K,C')
M1' = CBC-Dec(K,C')
(X,plen) <- lastbyte(M1')
For i = 0 to padlen do
    (X,plen') <- lastbyte(X)
    If plen' != plen return Error
Return Ok

Chosen-Ciphertext Attacks on CBC

Attack	Description	Year
Vaudenay	10's of chosen ciphertexts, recovers message bits from a ciphertext. Called "padding oracle attack"	2001
Canvel et al.	Shows how to use Vaudenay's ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against SSH	2009
Duong, Rizzo	Breaks ASP.net encryption	2011
Jager, Somorovsky	XML encryption standard	2011
Duong, Rizzo	"Beast" attacks against TLS	2011

Compress, then Encrypt?

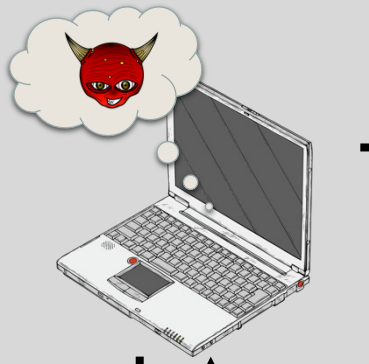


Second message compresses better than first:

network observer can see the difference in ciphertext sizes

Rizzo, Duong 2012

The CRIME Attack (Simplified)

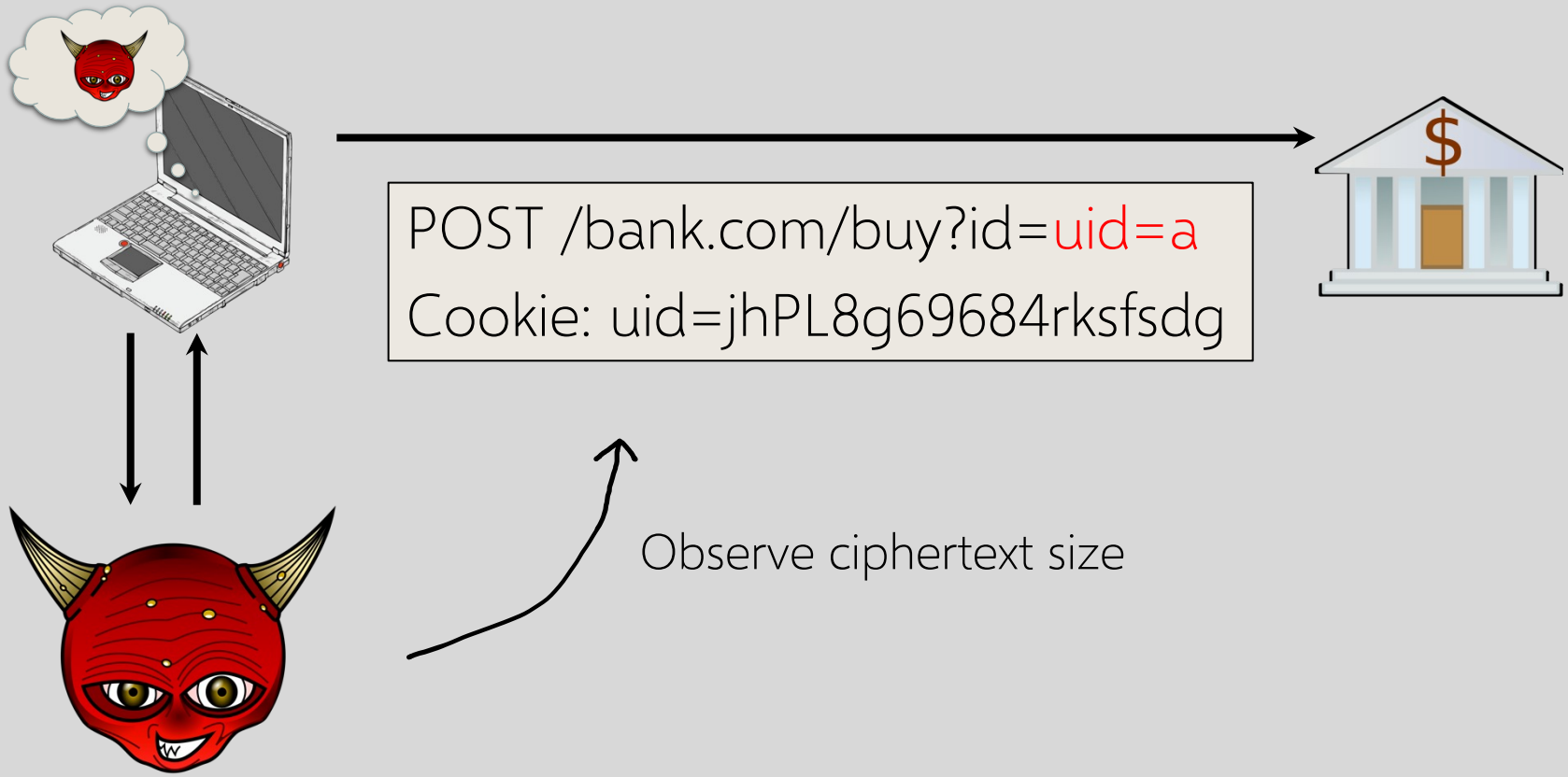


POST /bank.com/buy?id=aapl
Cookie: uid=jhPL8g69684rksfsdg

Malicious JavaScript can issue requests to the bank's website...
... but cannot read the cookie value

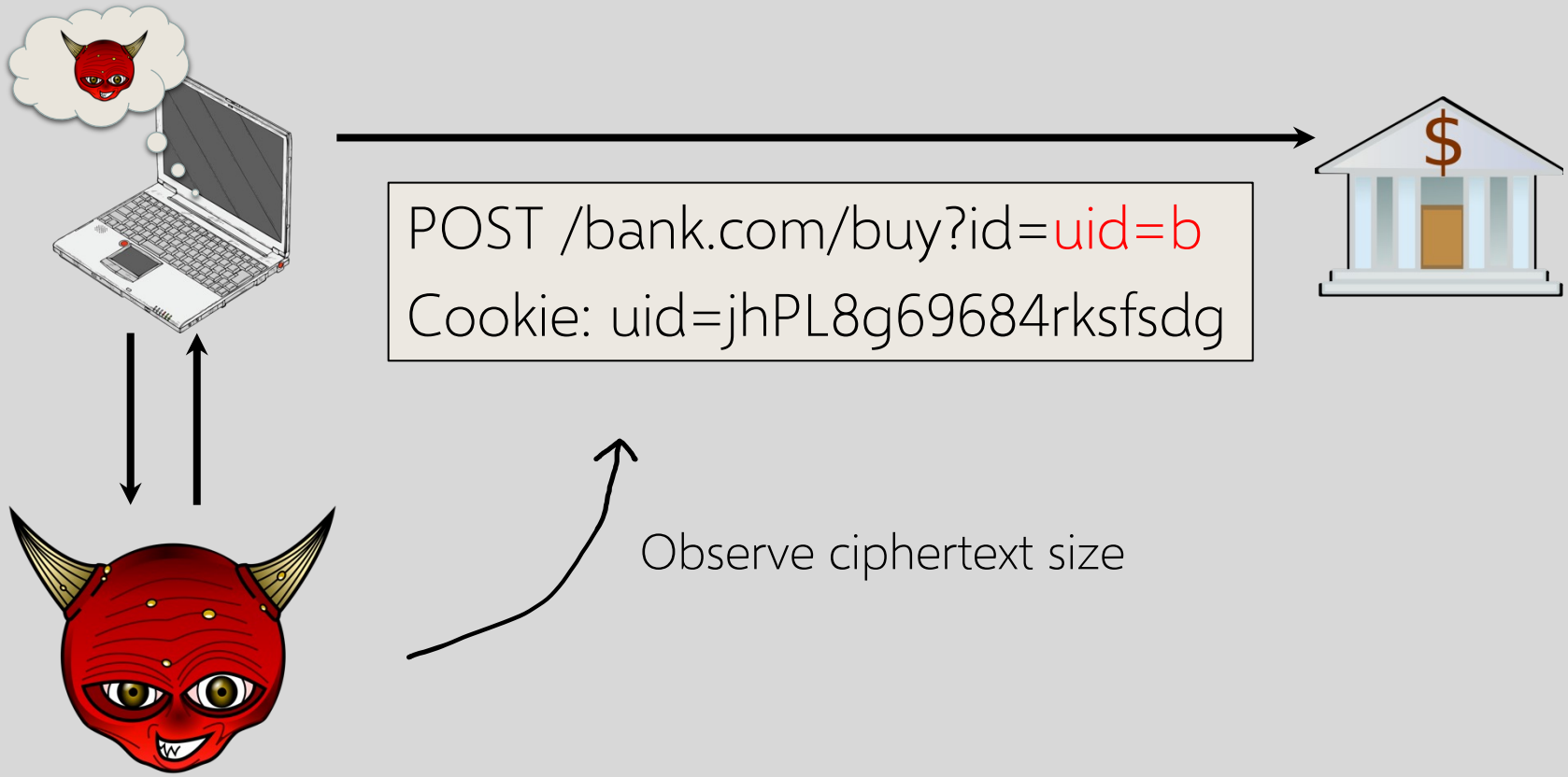
Rizzo, Duong 2012

The CRIME Attack (Simplified)



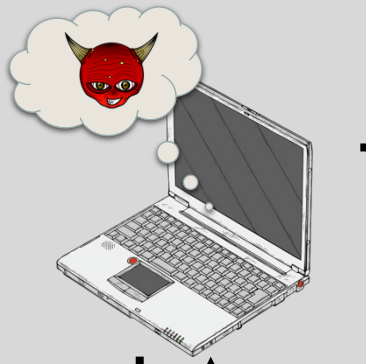
Rizzo, Duong 2012

The CRIME Attack (Simplified)



Rizzo, Duong 2012

The CRIME Attack (Simplified)



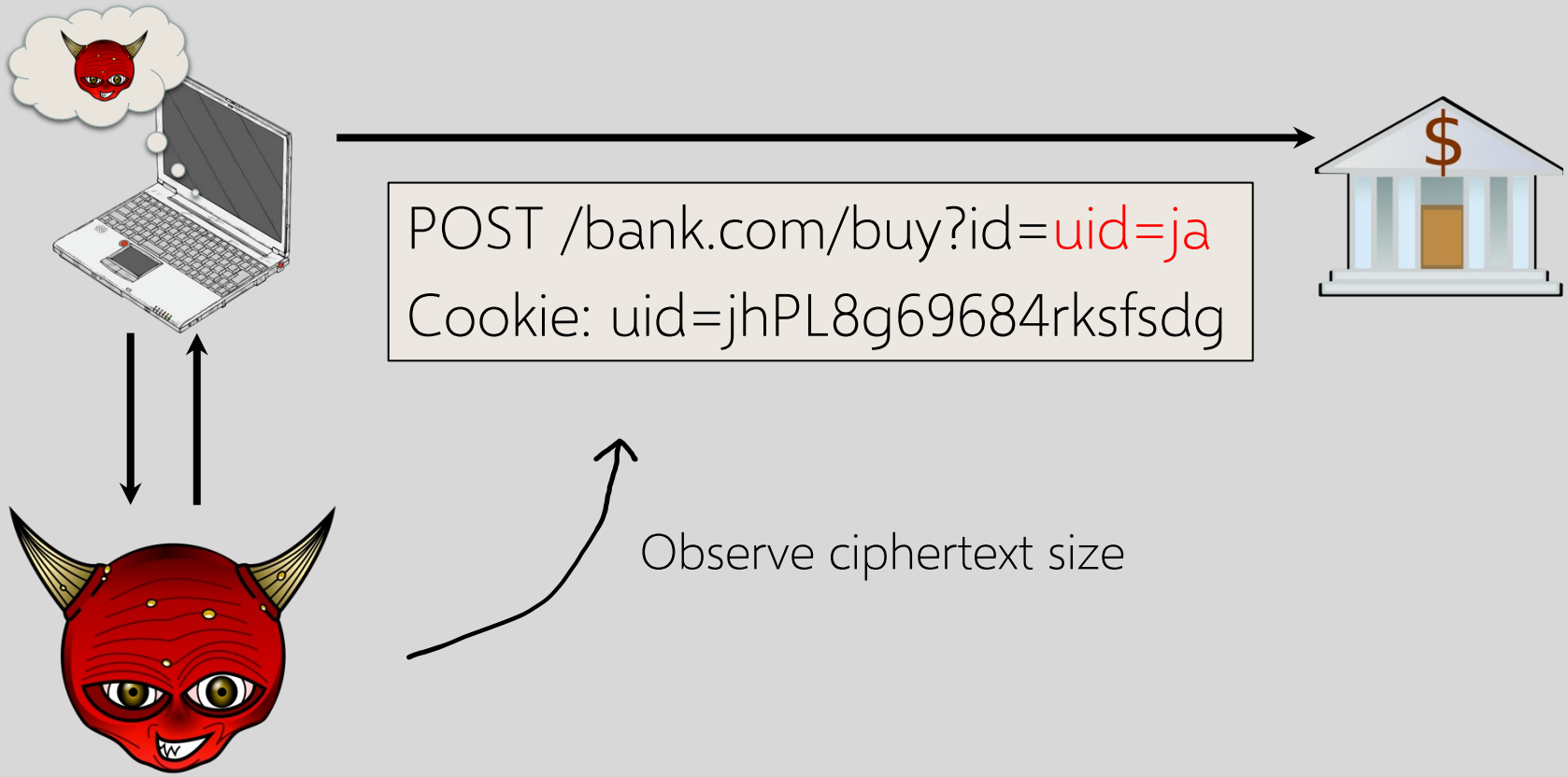
POST /bank.com/buy?id=uid=j
Cookie: uid=jhPL8g69684rksfsdg

Observe ciphertext size

Ciphertext slightly shorter! (due to compression)
First character of cookie is "j"

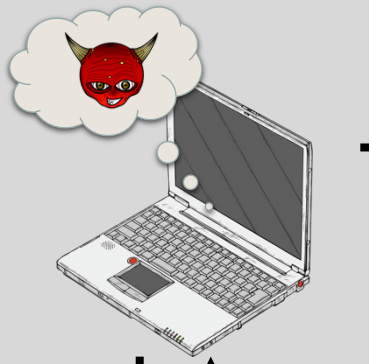
Rizzo, Duong 2012

The CRIME Attack (Simplified)



Rizzo, Duong 2012

The CRIME Attack (Simplified)



POST /bank.com/buy?id=uid=jh
Cookie: uid=jhPL8g69684rksfsdg

Observe ciphertext size

Ciphertext slightly shorter! (due to compression)
Second character of cookie is "h"

Recover entire cookie after 256 x |cookie| tries (in minutes)

Solutions

- Disable compression
- Use a different compression context for parts under JavaScript control and parts that are not
- Change the cookie after every request

😬

Does not eliminate leakage due to compression

Does not eliminate leakage due to compression

When Is an Encryption Scheme "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 System Be Attacked?

Attackers knows the ciphertext and the encryption algorithm

- What else does the attacker know? Depends on the application!

Known-plaintext attack (stronger)

- Attacker knows some plaintext-ciphertext pairs

Chosen-plaintext attack (even stronger)

- Attacker can obtain ciphertext for any plaintext of his choice

Chosen-ciphertext attack (very strong)

- Attacker can decrypt any ciphertext except the target
- Sometimes very realistic

OpenSSL
Cryptography and SSL/TLS Toolkit

Known-Plaintext Attack

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

From Kevin Mitnick's "The Art of Intrusion"

Very Informal Intuition

Minimum security requirement for any modern encryption scheme

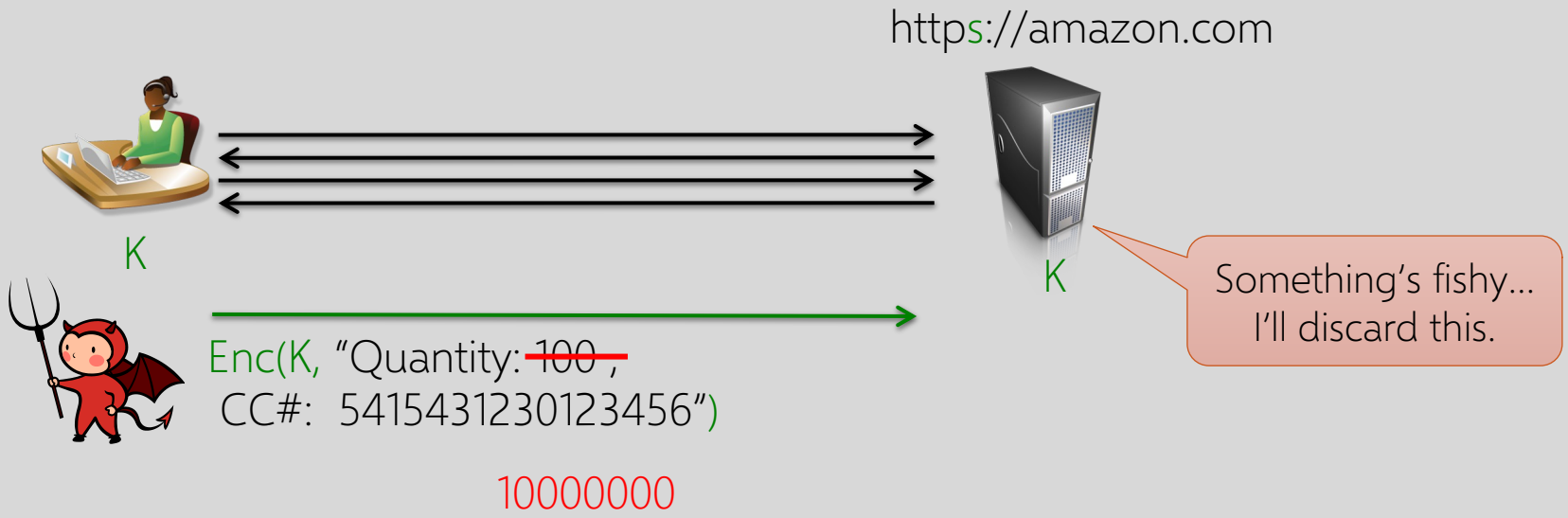
- Security against chosen-plaintext attack
  - Ciphertext leaks no information about the plaintext
  - Even if the attacker correctly guesses the plaintext, he cannot verify his guess
  - Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
- Security against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

Simple exercise: show that ECB mode is not secure

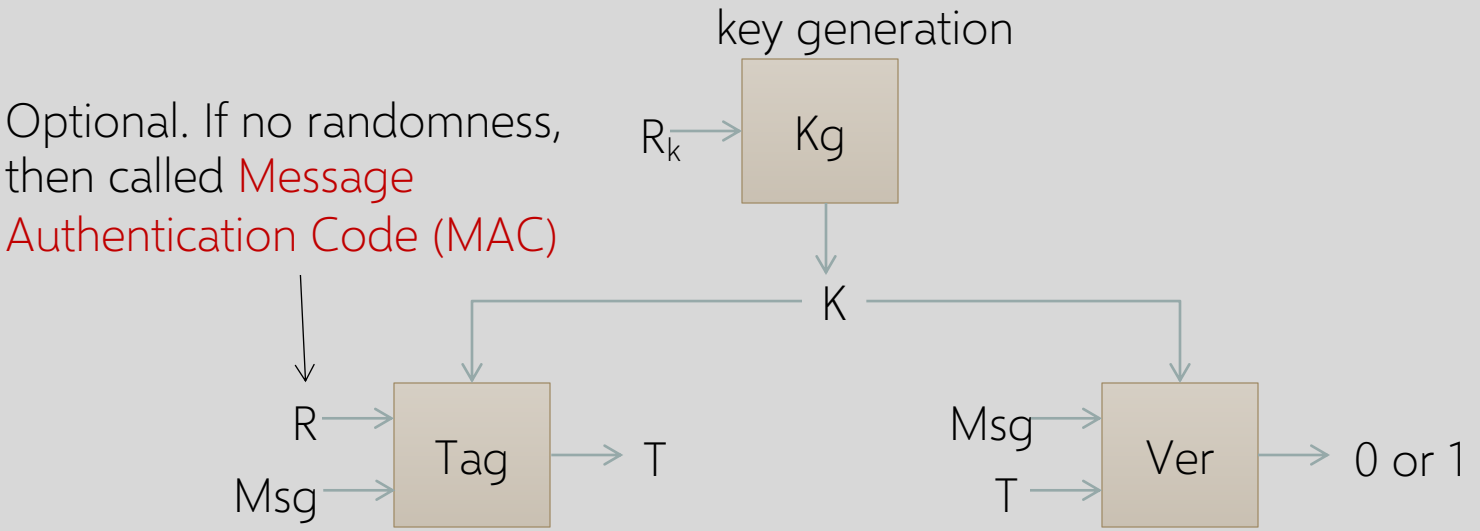
Authenticated Encryption

Goal: Hide message (confidentiality) and detect tampering (integrity)

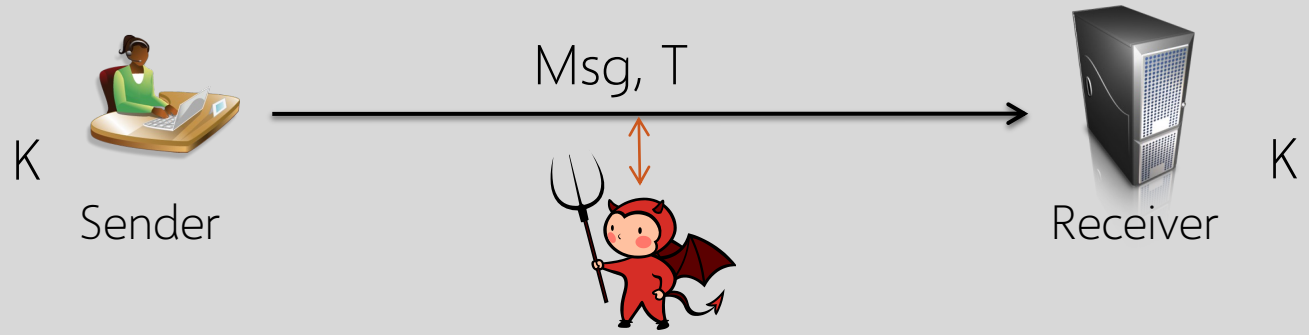
Can build by combining encryption with message authentication scheme



Message Authentication



Message Authentication



Two algorithms:
Tag(K,Msg) outputs a tag T
Verify(K,Msg,T) outputs 0/1 (invalid / valid)

Security: No computationally efficient attacker can forge tags for a new message even when attacker gets

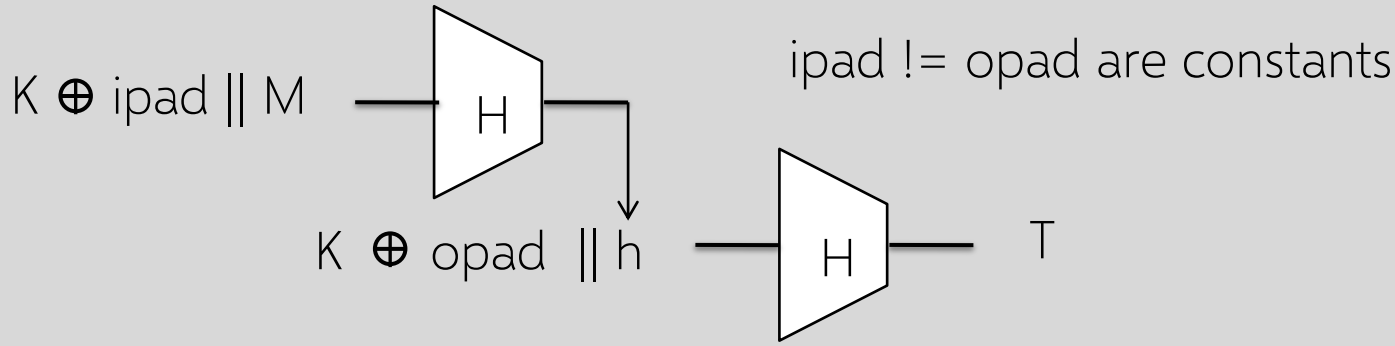
$(Msg_1 , T_1) , (Msg_2, T_2), \ldots , (Msg_q , T_q)$

for messages of his choosing and reasonably large q

Message Authentication with HMAC

Kgen outputs uniform bit string K

Tag(K,M) = HMAC(K,M) defined by:

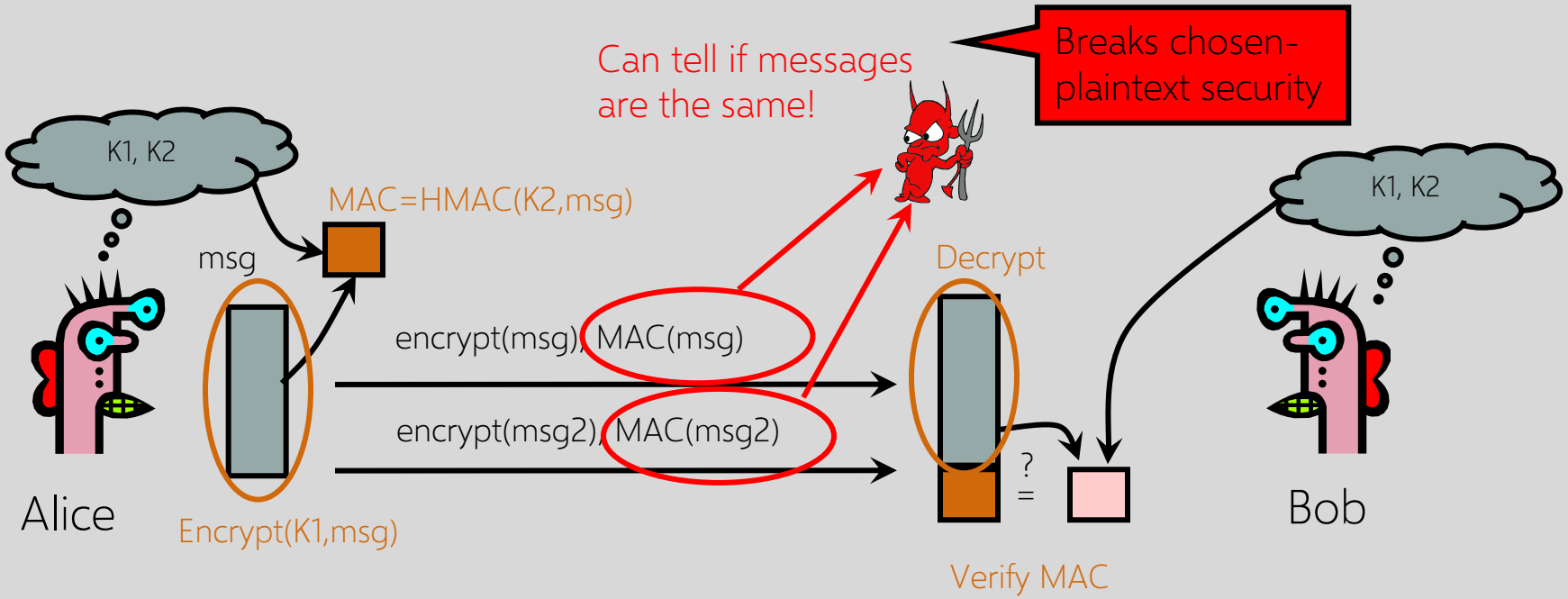


To verify a M,T pair, check if HMAC(K,M) = T

Unforgeability holds if hash function H behaves like a random function

Goal: confidentiality + integrity + authentication

Encrypt + MAC



MAC is deterministic: messages are equal ⇒ their MACs are equal

Solution: Encrypt, then MAC (IPsec, TLS 1.3)
MAC, then encrypt (SSL)

Authenticated Encryption Schemes

Attack	Inventor(s)	Notes
OCB (Offset Codebook)	Rogaway	One-pass mode and fastest
AES-GCM (Galois Counter Mode)	McGrew, Viega	CTR mode plus Carter-Wegman MAC
ChaCha20/Poly1305	Bernstein	"essentially" CTR mode plus special Carter-Wegman MAC
CCM	Housley, Ferguson, Whiting	CTR mode plus CBC-MAC
EAX	Wagner, Bellare, Rogaway	CTR mode plus OMAC

Other considerations in authenticated encryption (AE):
robustness & IV misuse, deterministic AE, associated data, …