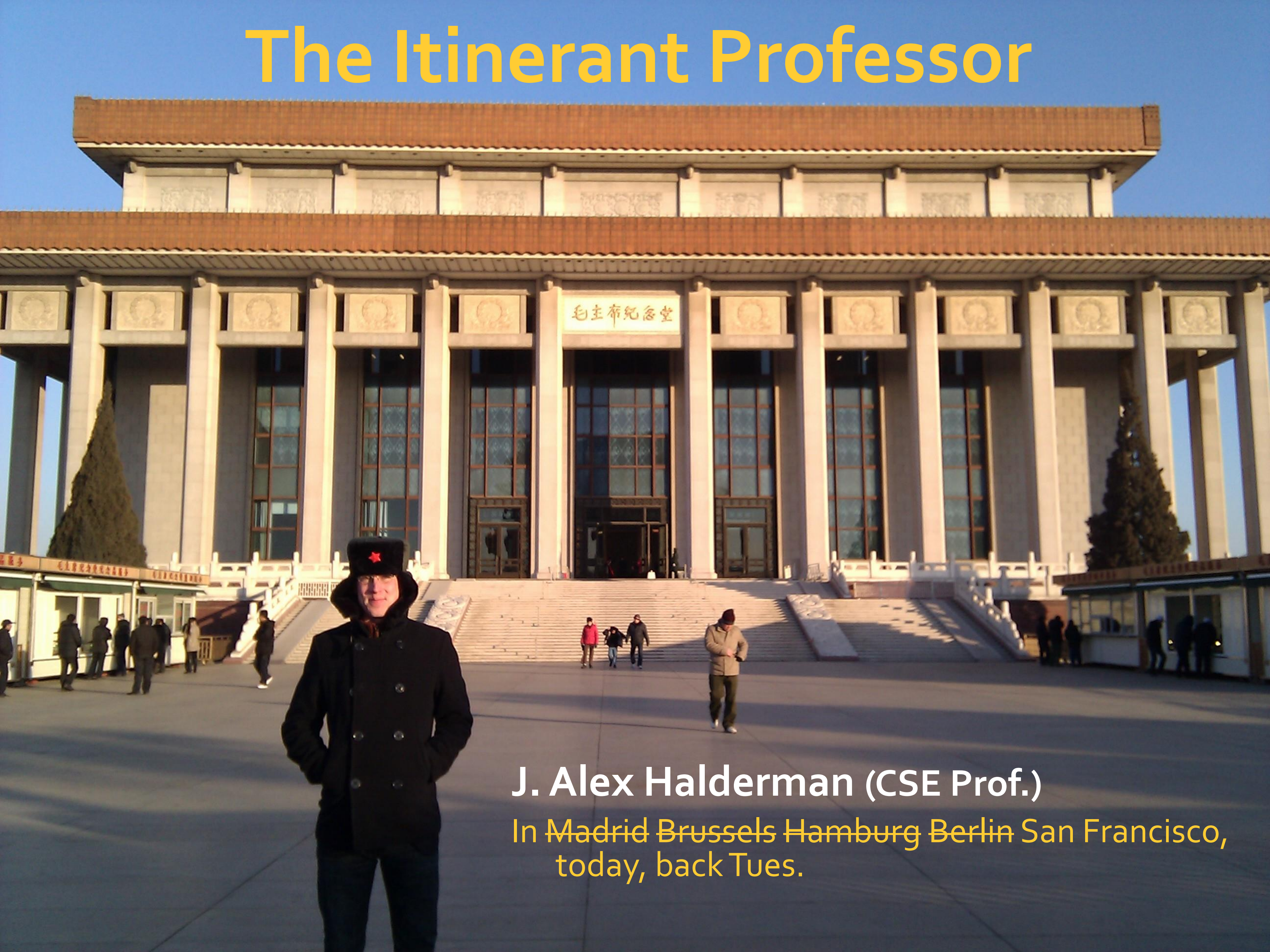
Today's Lecture: Crypto Crash-Course

EECS 588: Computer and Network Security
January 7, 2016

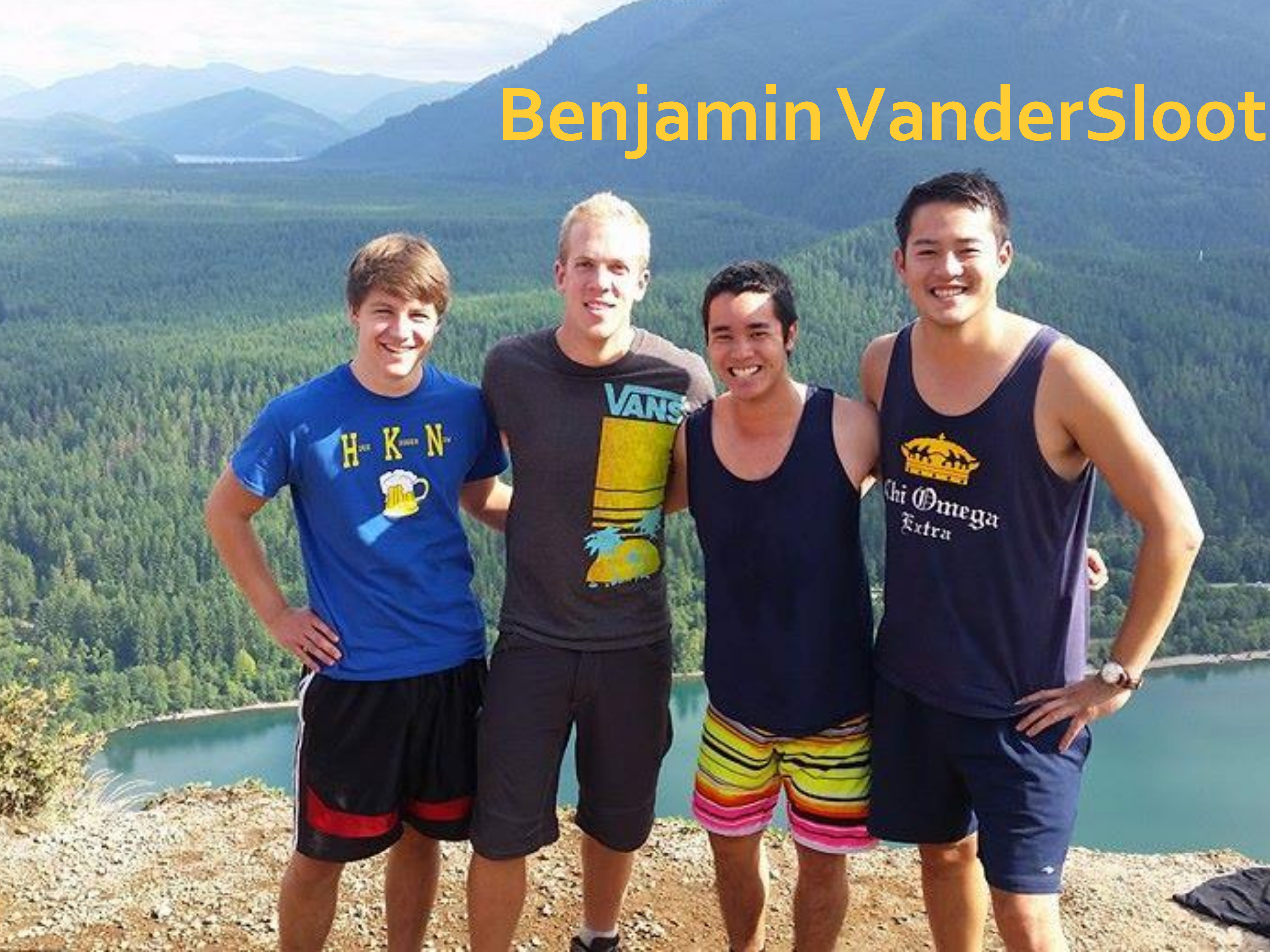
The Itinerant Professor



J. Alex Halderman (CSE Prof.)

In ~~Madrid~~ ~~Brussels~~ ~~Hamburg~~ ~~Berlin~~ San Francisco,
today, back Tues.

Benjamin VanderSloot



Goals for this Course

- Gain hands-on experience
  - Building secure systems
  - Evaluating system security
- Prepare for research
  - Computer security subfield
  - Security-related issues in other areas
- Generally, improve research and communication skills
- Learn to be a `1337 hax0r`, but an ethical one!

Building Blocks
The security mindset, thinking like an attacker, reasoning about risk, research ethics
Symmetric ciphers, hash functions, message authentication codes, pseudorandom generators
Key exchange, public-key cryptography, key management, the TLS protocol

Software Security
Exploitable bugs: buffer overflows and other common vulnerabilities – attacks and defenses
Malware: viruses, spyware, rootkits – operation and detection
Automated security testing and tools for writing secure code
Virtualization, sandboxing, and OS-level defenses

Web Security
The browser security model
Web site attacks and defenses: cross-site scripting, SQL injection, cross-site reference forgery
Internet crime: spam, phishing, botnets – technical and nontechnical responses

Network Security
Network protocols security: TCP and DNS – attacks and defenses
Policing packets: Firewalls, VPNs, intrusion detection
Denial of service attacks and defenses
Data privacy, anonymity, censorship, surveillance

Advanced Topics
Hardware security – attacks and defenses
Trusted computing and digital rights management
Electronic voting – vulnerabilities, cryptographic voting protocols

Not a crypto course

Getting a Seat

- Long waitlist, but odds are good.

Communication

Course Web Site
https://eecs588.org
announcements, schedule, readings

Email Us
jhalderm@umich.edu
eecs588@umich.edu
suggestions, questions, concerns

Today’s Class

Essential Cryptography

- The Cryptographer’s View
- Hash Functions
- Message-Authentication Codes
- Generating Random Numbers
- Block Ciphers

Basic Cryptography Problems

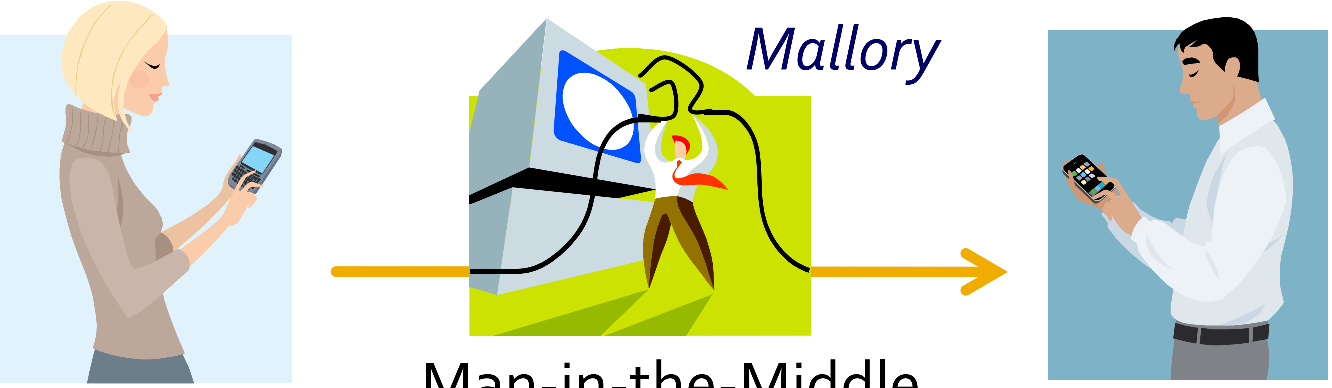
Message

Eve

Alice

Bob

Passive Eavesdropper



Mallory

Man-in-the-Middle

Ingredients for a Secure Channel

Confidentiality

Attacker can't see the message

Symmetric Ciphers

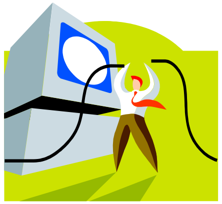
Eve

Integrity

Attacker can't modify the message

Message Authentication Codes (MACs)

Mallory



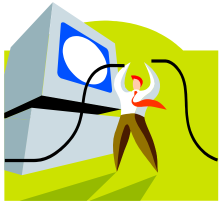
Ingredients for a Secure Channel

Authentication

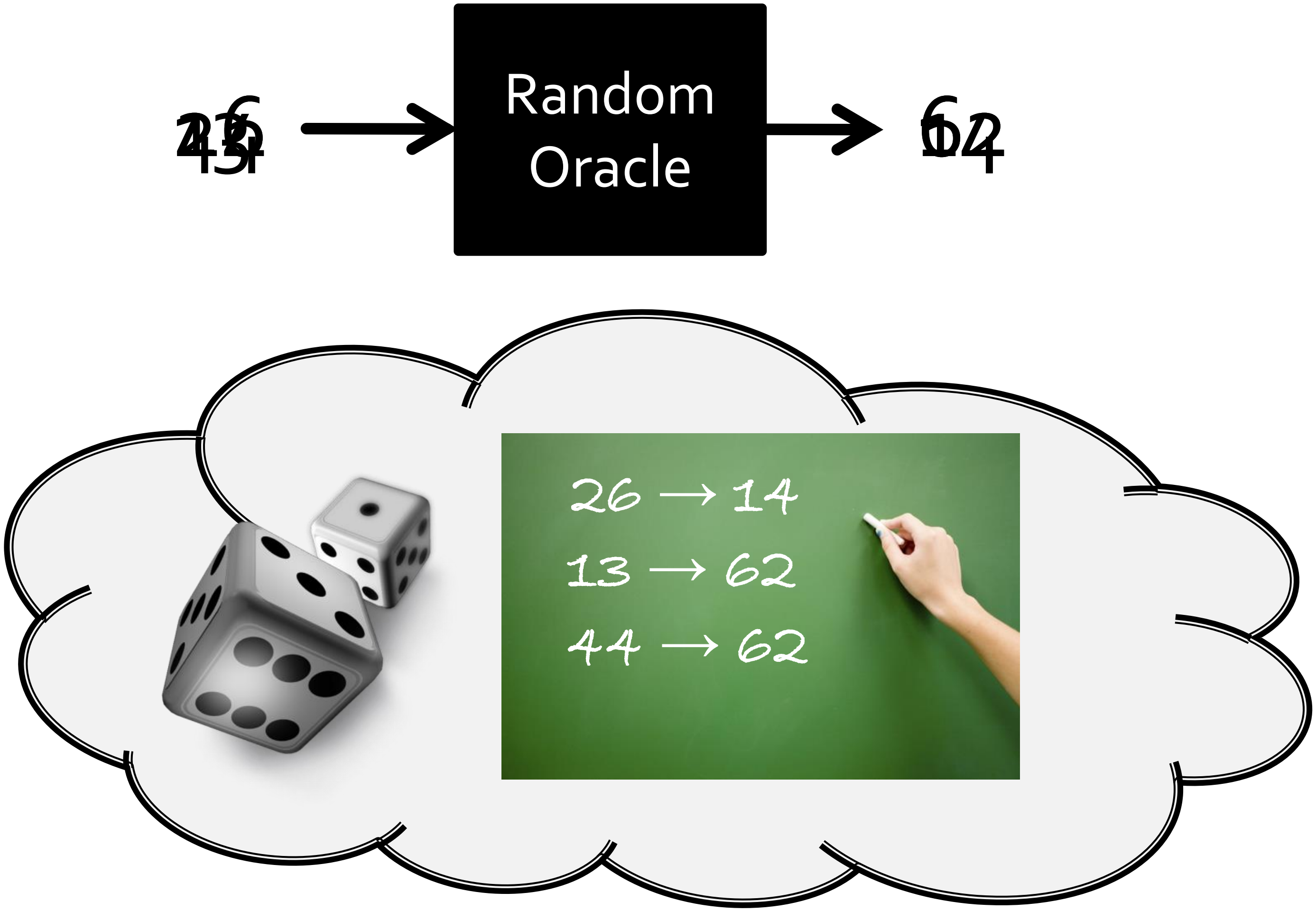
Attacker can’t impersonate the recipient

Public-Key Cryptography

Mallory



The Cryptographer's View



26 → 14

13 → 62

44 → 62

Practical Random Oracles?

Suppose domain is size 2^{256}…

Pseudorandom Functions (PRFs)

(A function randomly chosen from a *family* of PRFs is computationally indistinguishable from a Random Oracle)

≈ Message Authentication Codes (MACs)

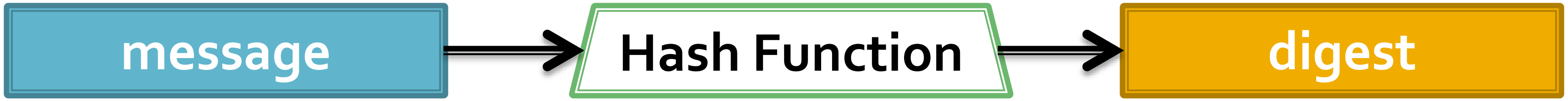
Pseudorandom Permutations

≈ Symmetric Ciphers

Hash Functions

- Ideal: Random mapping from *any input* to a *set of output*

- Caution! Real hashes don't match our ideal



message → Hash Function → digest

Ideal Hash Function

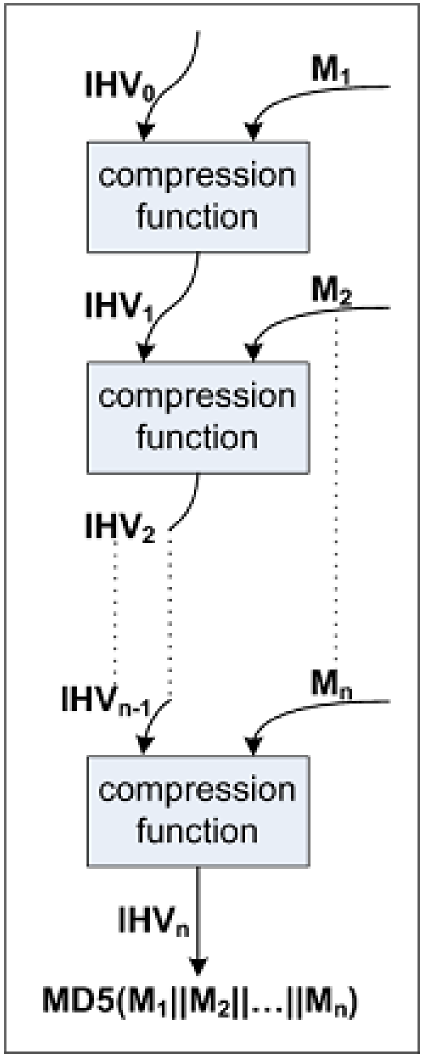
1. Easy to compute H*(m)* for all *m*
2. Infeasible to compute *m* from H(*m*)
3. Infeasible to modify *m* without changing H(*m*)
4. Infeasible to find two messages with the same hash

Hash Function Requirements

- First pre-image resistance
  - Given h(x), cannot find x
- Second pre-image resistance
  - Given m_1, cannot find m_2 s.t. $h(m_1) = h(m_2)$
- Collision resistance
  - Given nothing, find *any* m_1 != m_2 s.t. $h(m_1) = h(m_2)$
  - Birthday Attack

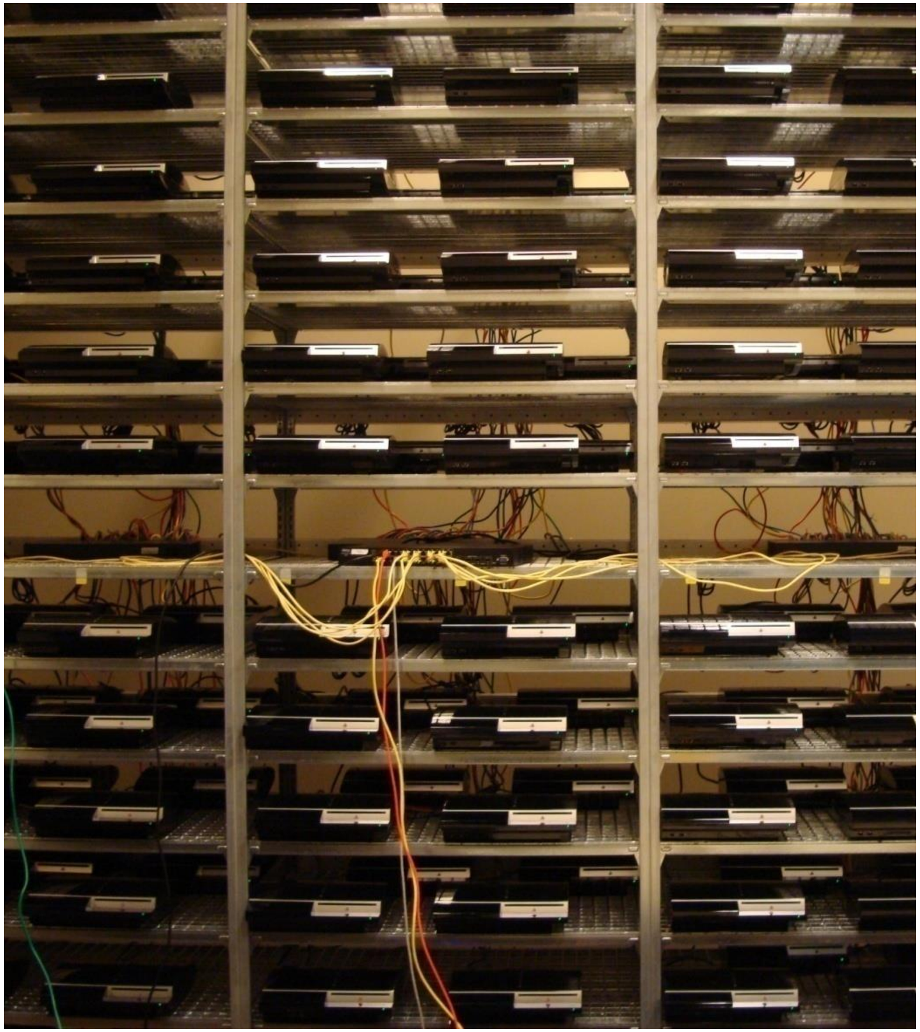
MD5 Hash Function

- Designed in 1992 by Ron Rivest
  - 128-bit output
  - 128-bit internal state
  - 512-bit block size
- Like most hash functions, uses block-chaining construction



MD5 is Unsafe – Never use it!

- First flaws in 1996; by 2007, researchers demonstrated a collision
- Chaining allows chosen prefix attack
- Dec. 2008: others used this to fake SSL certificates (cluster of 200 PS3s)



MD5 Collision

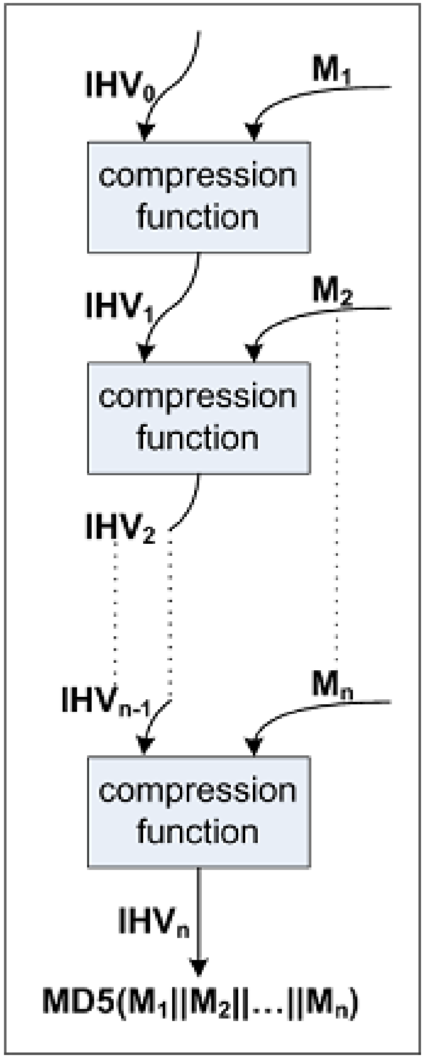
```
d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70

d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70
```

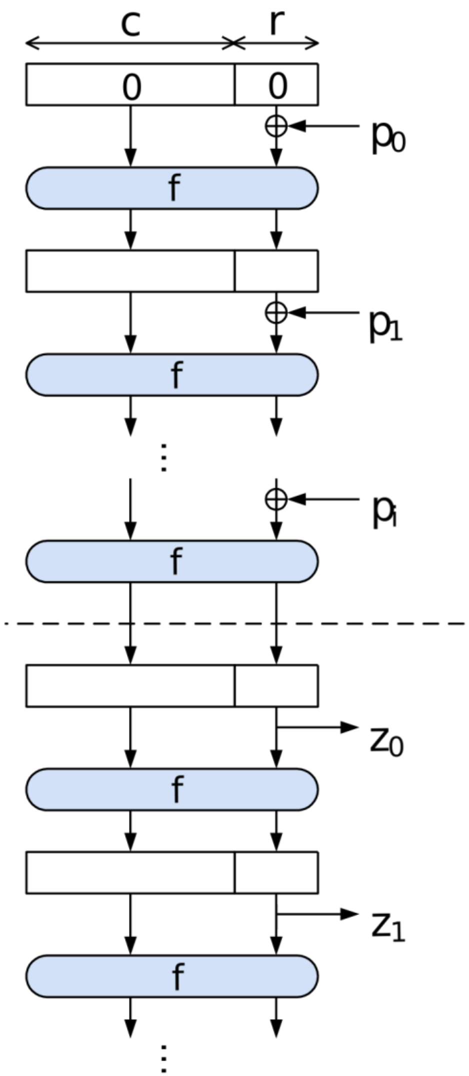
Both of these blocks hash to 79054025255fb1a26e4bc422aef54eb4

SHA Hash Functions

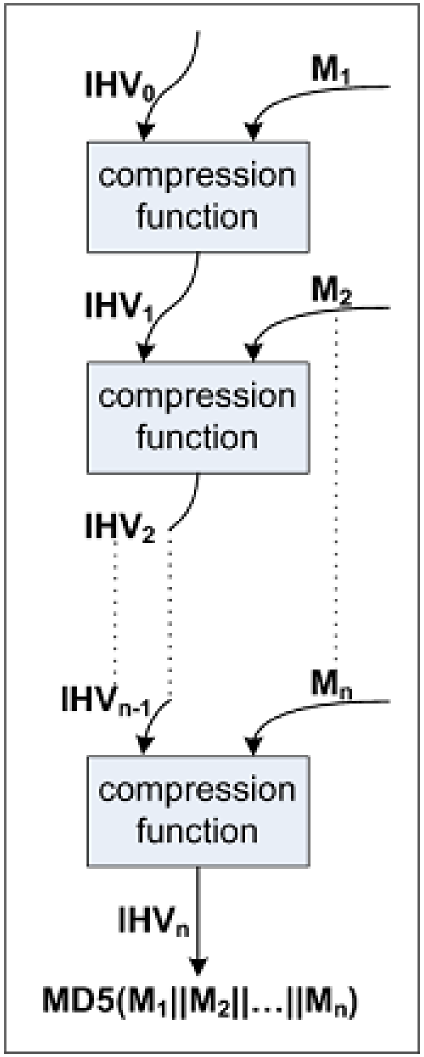
- SHA-1 – standardized by NIST in 1995
  - 160-bit output and internal state
  - 512-bit block size
- SHA-2 – extension published in 2001
  - 256 (or 512)-bit output and internal state
  - 512 (or 1024)-bit block size
- SHA-3 – chosen by NIST in 2012
  - 256 (512)-bit output
  - Different “sponge” construction

Block chaining vs. Sponge-construction





Tricky! Length Extension Attacks



Given hash of secret *x*, trivial to find hash of *x* || *p* || *m* for padding *p* and arbitrary *m*

Block chaining hashes are vulnerable!

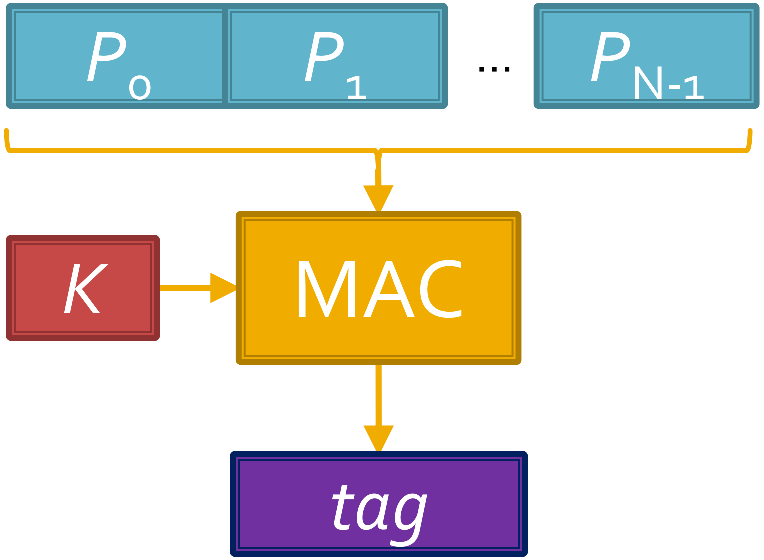
Is SHA-1 Safe?

- Significant cryptanalysis since 2005
- Improved attacks show complexity of finding a collision < 2^{51}(ideally security would be 2^{80} – why?)
- Attacks only get better …
- The SHAppening
  - Freestart collision found

- **Use SHA-256**

Message Authentication Codes

- **Prevents tampering with messages.** Like a *family* of pseudorandom functions, with a key to select among them



Construction: HMAC

Given a hash function H:

HMAC(K,m) = H(($K \oplus$ pad1) || H($K \oplus$ pad2 || m))
for constants pad1 and pad2

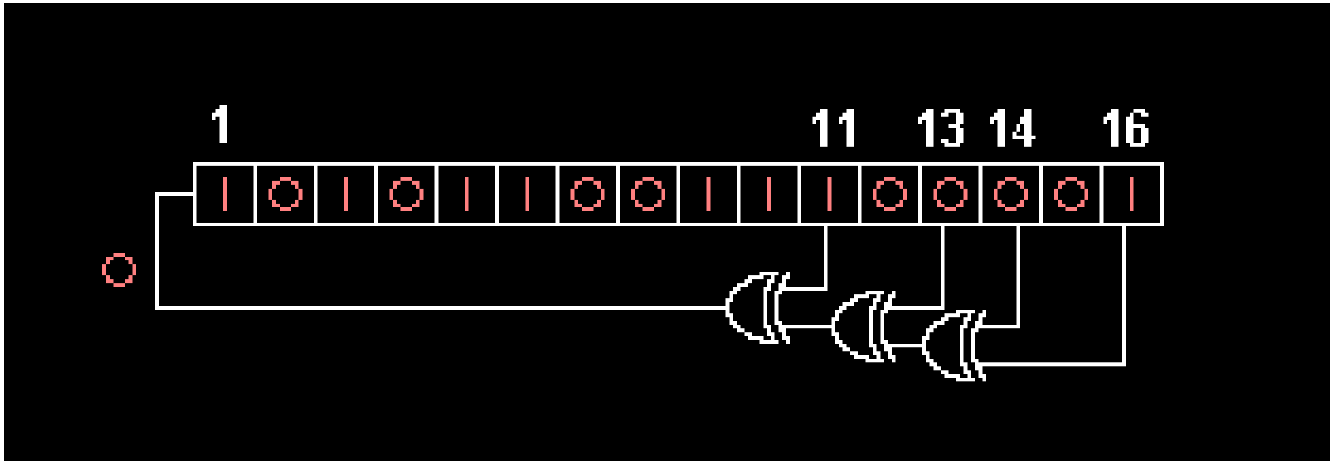
Provides nice provable security properties

What Should You Use?

- **Use HMAC-SHA256**
  - Use a constant key to get a length-extension resistant hash function

Generating Random Numbers

- What’s wrong with srand() and rand()?



Generating Random Numbers

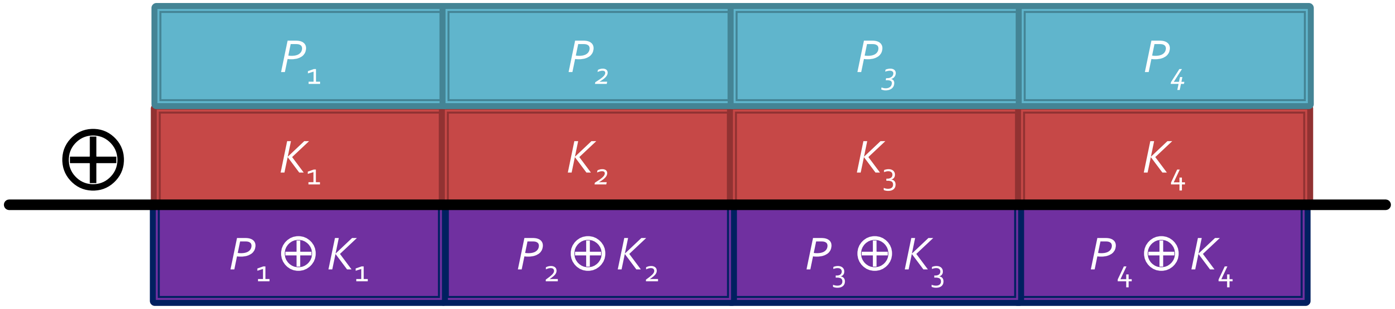
- What’s wrong with srand() and rand()?
- Why not use a secure hash?
  - “Cryptographic Pseudorandom Number Generator” (CPRNG)
- Tricky details…
  - Seeding with true randomness (“entropy”)
  - Forward secrecy
- Most OSes do the hard work for you*
  - On Linux, use `/dev/random` and `/dev/urandom`

One-Time Pads

Provably secure encryption…

… that often fails in practice.

One-Time Pads



P_1 | P_2 | P_3 | P_4

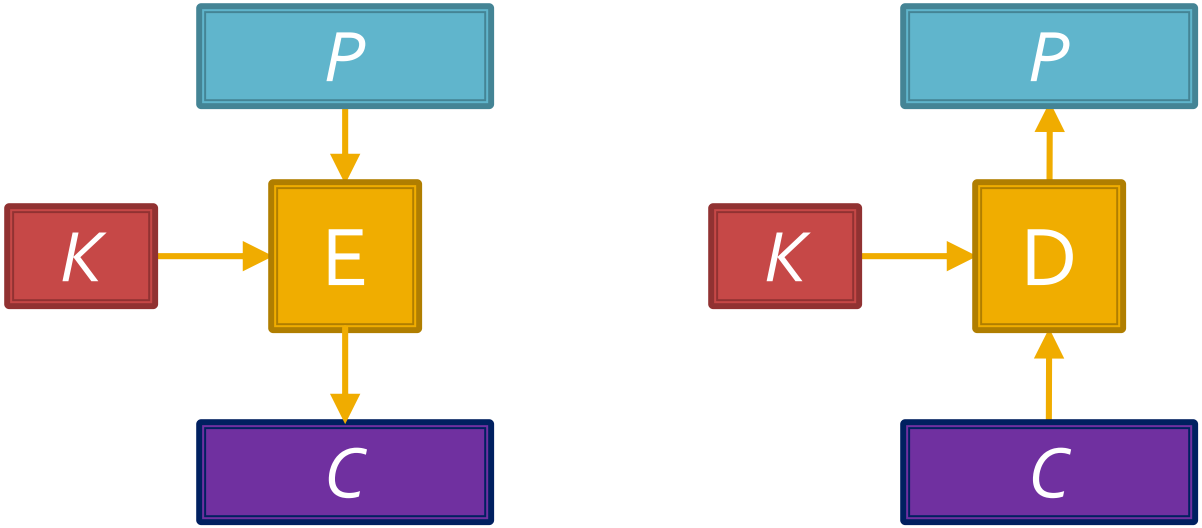
$\oplus$ K_1 | K_2 | K_3 | K_4

$P_1 \oplus K_1$ | $P_2 \oplus K_2$ | $P_3 \oplus K_3$ | $P_4 \oplus K_4$

$P_i \oplus K_i$	P_i	K_i
0	0	0
0	1	1
1	0	1
1	1	0

Block Ciphers

- Ideal block cipher:
  Like a *family* of pseudorandom *permutations* with a key to select among them

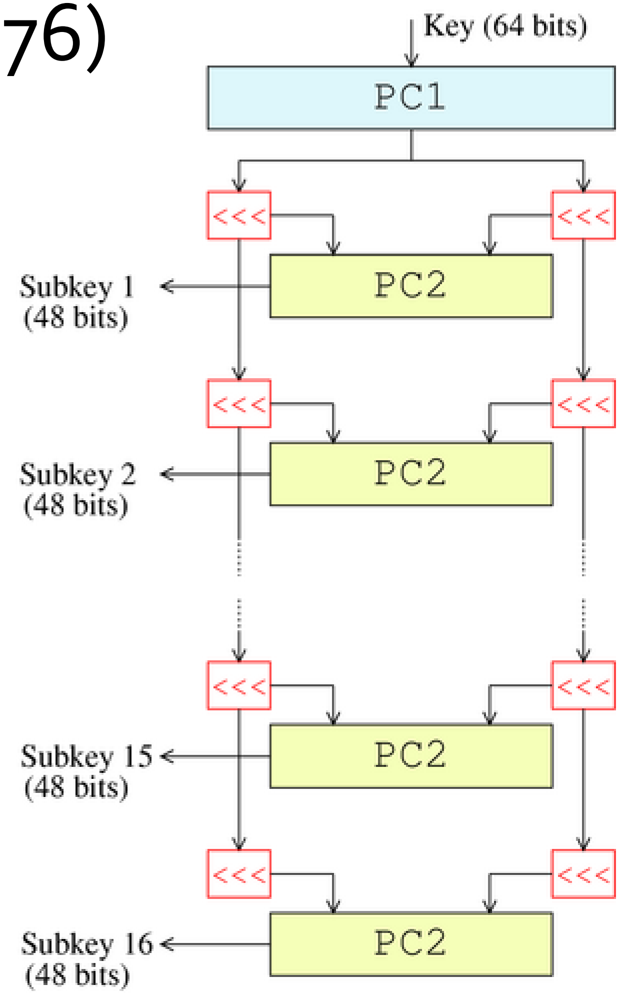


DES—Data Encryption Standard

- US Government standard (1976)
- Designed by IBM
  Tweaked by NSA

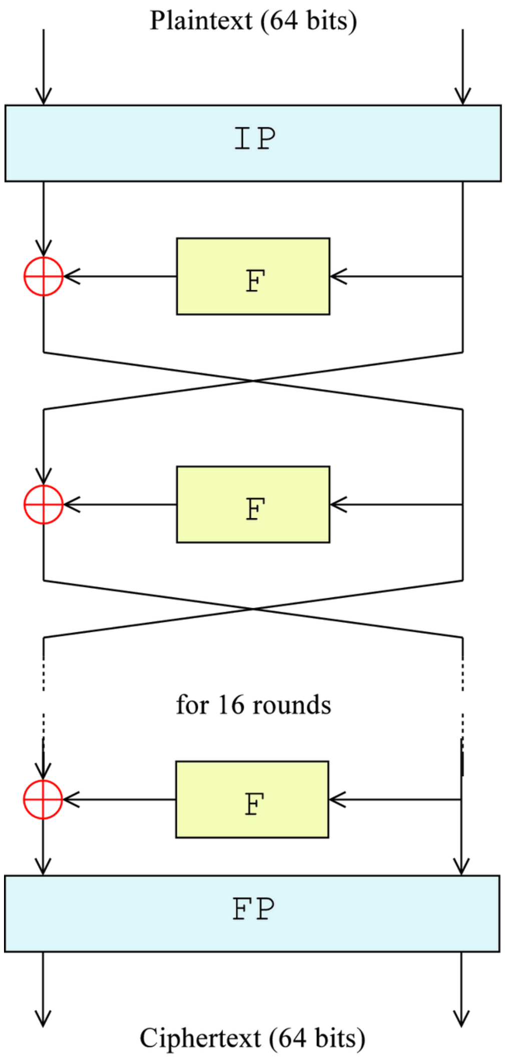
- 56-bit *key*
- 64-bit *blocks*
- 16 *rounds*

- Key schedule function generates 16 round keys:



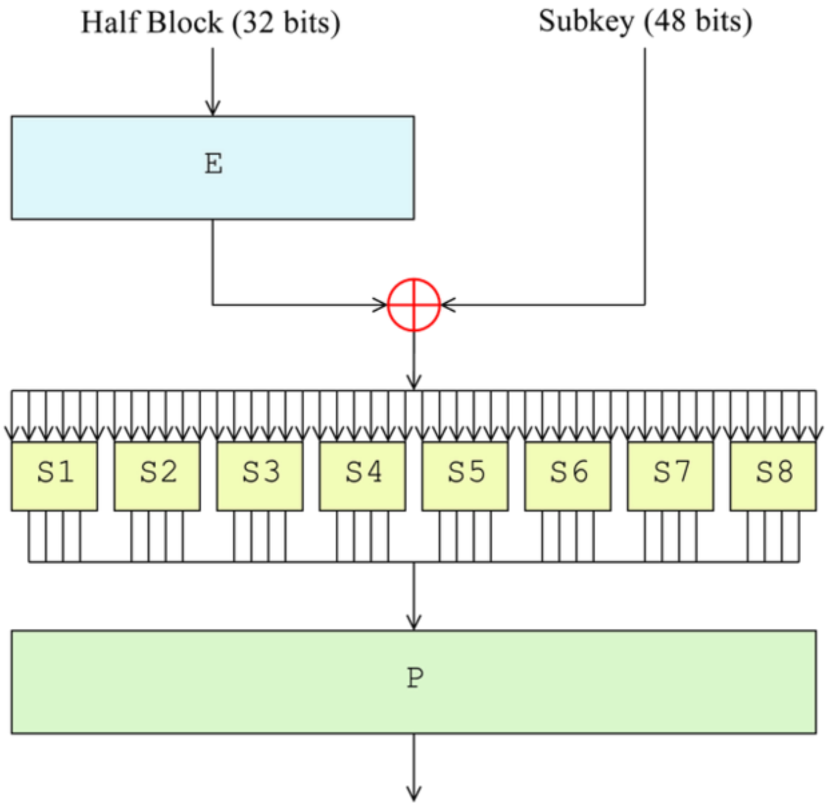
DES Encryption

- Feistel network
  - common block cipher construction
  - Each round uses the same Feistel function *F* (by itself a weak block cipher)
  - makes encryption and decryption symmetric—just reverse order of round keys



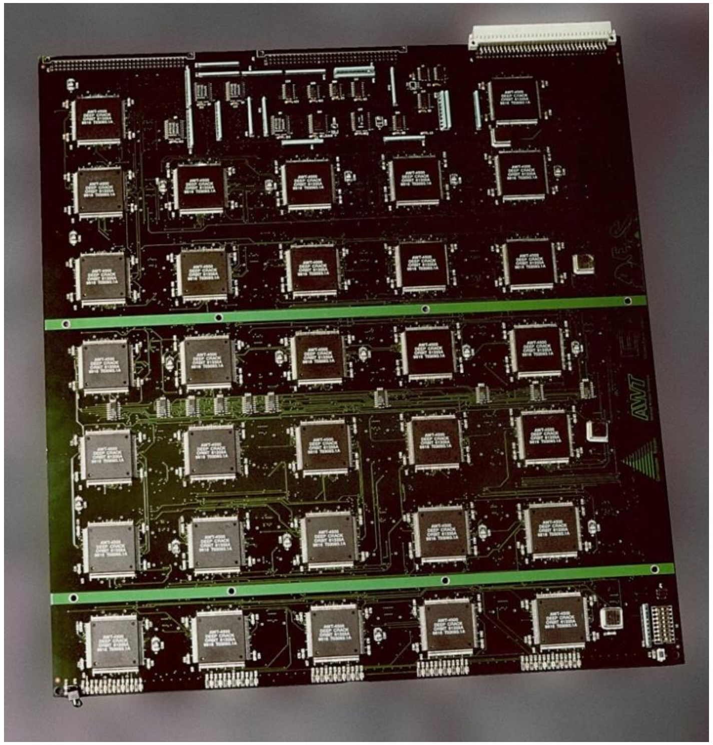
DES Feistel Function

- In each round:
  - Expansion Permutation *E*
    32 → 48 bits
  - S-boxes ("substitution")
    replace 6-bit values
  - Fixed Permutation P
    rearrange the 32 bits



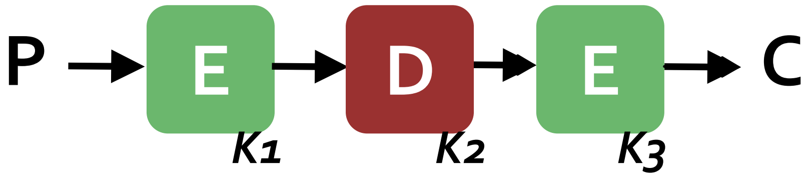
DES is Unsafe – Don’t Use It!

- Design has known weaknesses
- 56-bit key ***way*** too short
- EFF’s “Deep Crack” machine can brute force in 56 hours using FPGAs ($250k in 1998, far cheaper today)



3DES

- $E_{K_1, K_2, K_3}(P) = E_{K_3}(D_{K_2}(E_{K_1}(P)))$

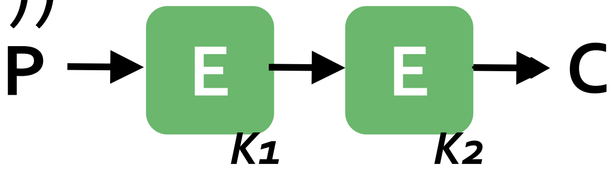


- Key options:
  - Option 1: independent keys (56*3 = 168 bit key)
  - Option 2: $K_1 = K_3$ (56*2 = 112 bit key)
  - Option 3: $K_1 = K_2 = K_3$ (Backward-compatible DES)

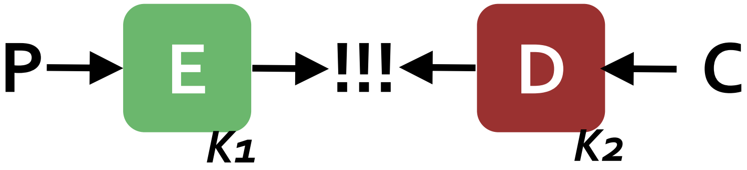
- What happened to 2DES?

2DES: Meet-in-the-middle attack

- "2DES": $E_{K_1, K_2}(P) = E_{K_2}(E_{K_1}(P))$



- Given P and C = $E_{K_2}(E_{K_1}(P))$, find both keys
  - For all K, generate $E_K(P)$ and $D_K(C)$
  - Find a match where $D_{K_2}(C) == E_{K_1}(P)$



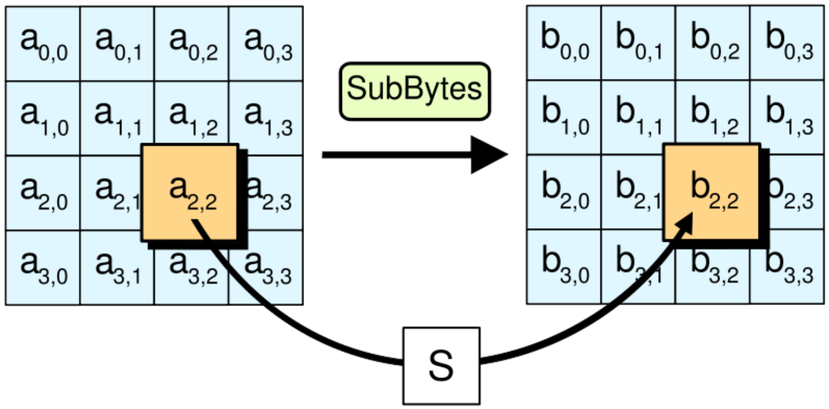
AES—Advanced Encryption Standard

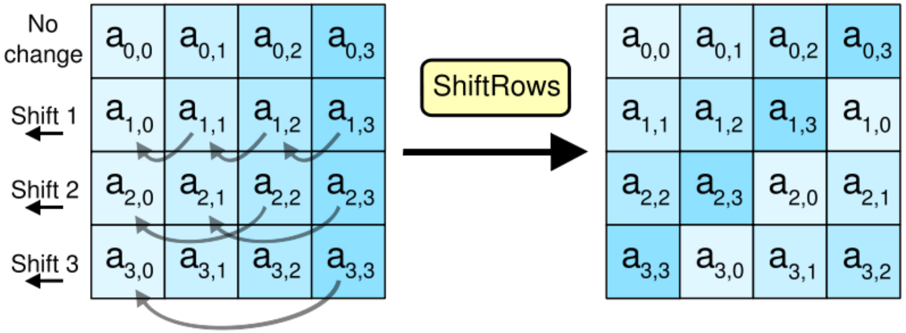
- Standardized by NIST in 2001 following open design competition (a.k.a. Rijndael)

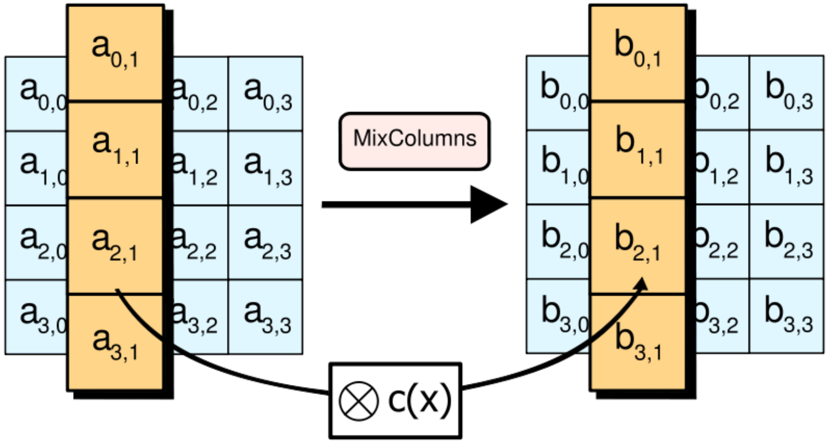
- 128-, 192-, or 256-bit key
- 128-bit blocks
- 10, 12, or 14 rounds

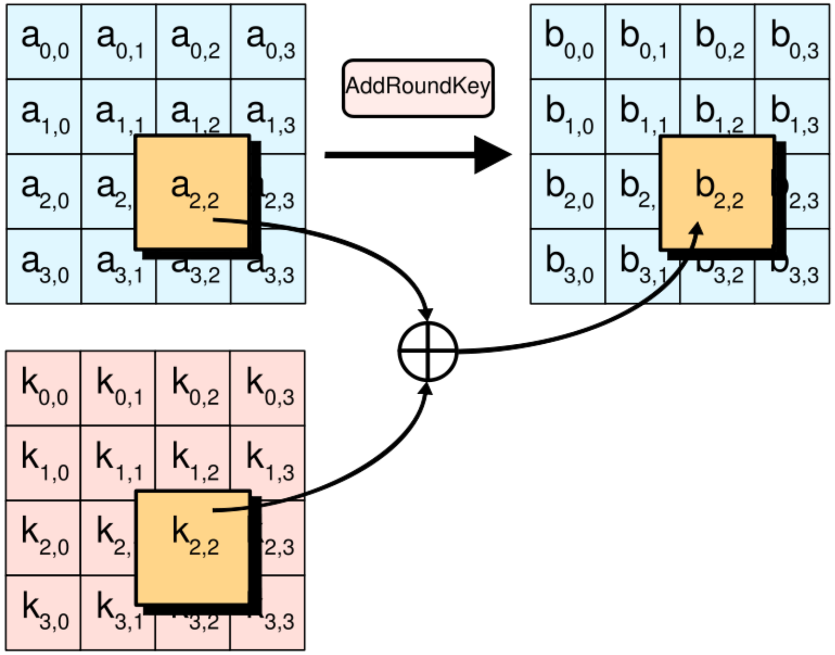
- Not a Feistel-network construction

One round of AES-128


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes$ c(x)
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

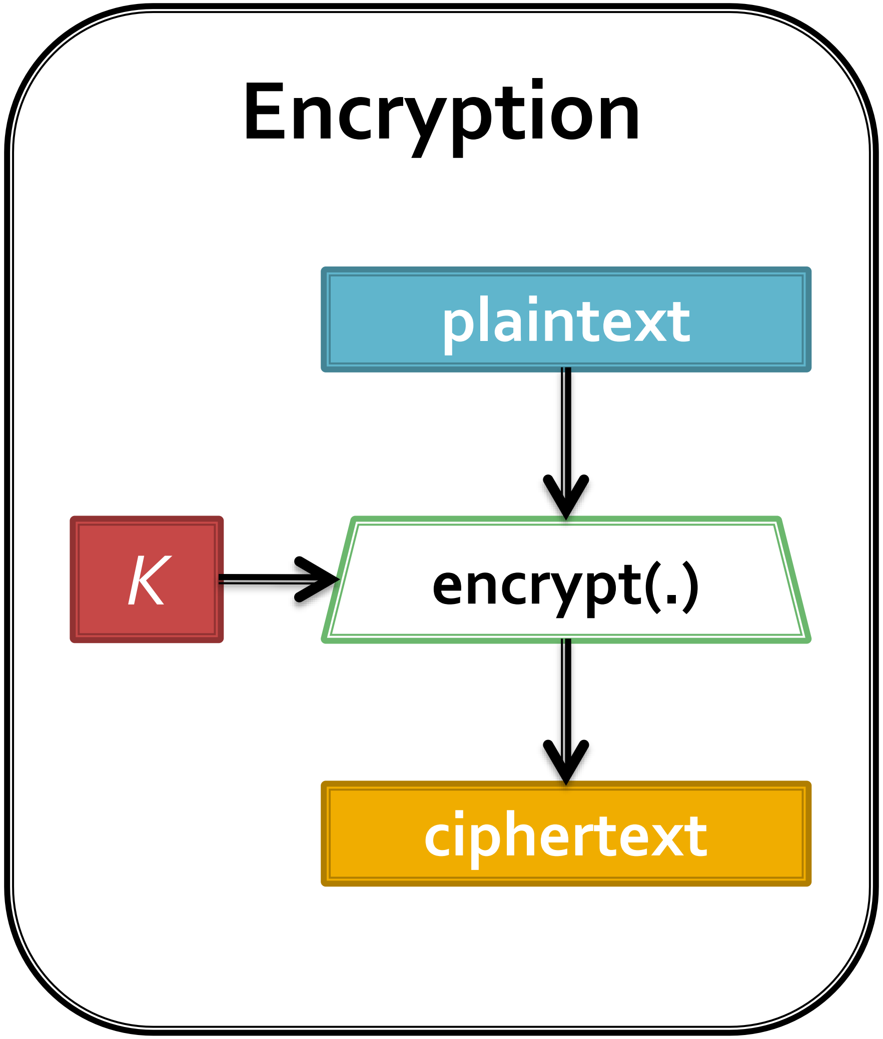

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$\oplus$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

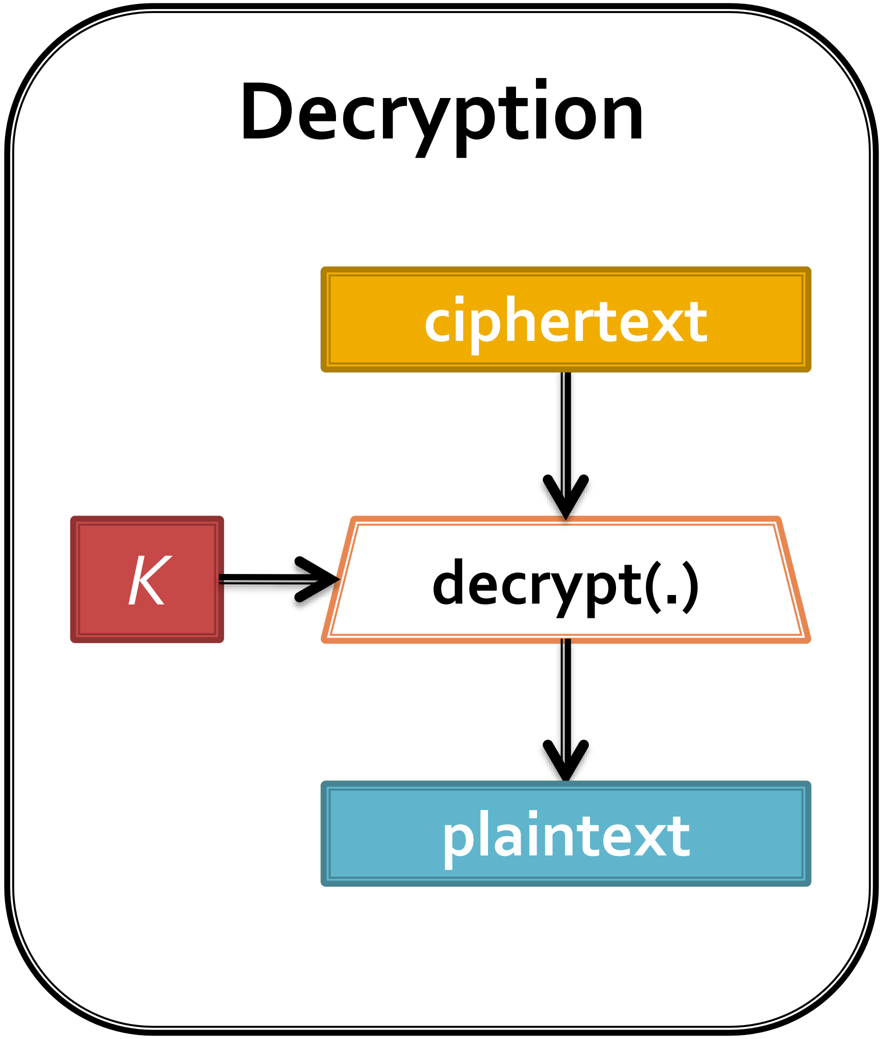
How Safe is AES?

- Known attacks against 128-bit AES if reduced to 7 rounds (instead of 10)
- 128-bit AES very widely used,
  though NSA requires 192- or 256-bit keys for SECRET and TOP SECRET data

- What should you use?
  - Conservative answer: Use 256-bit AES

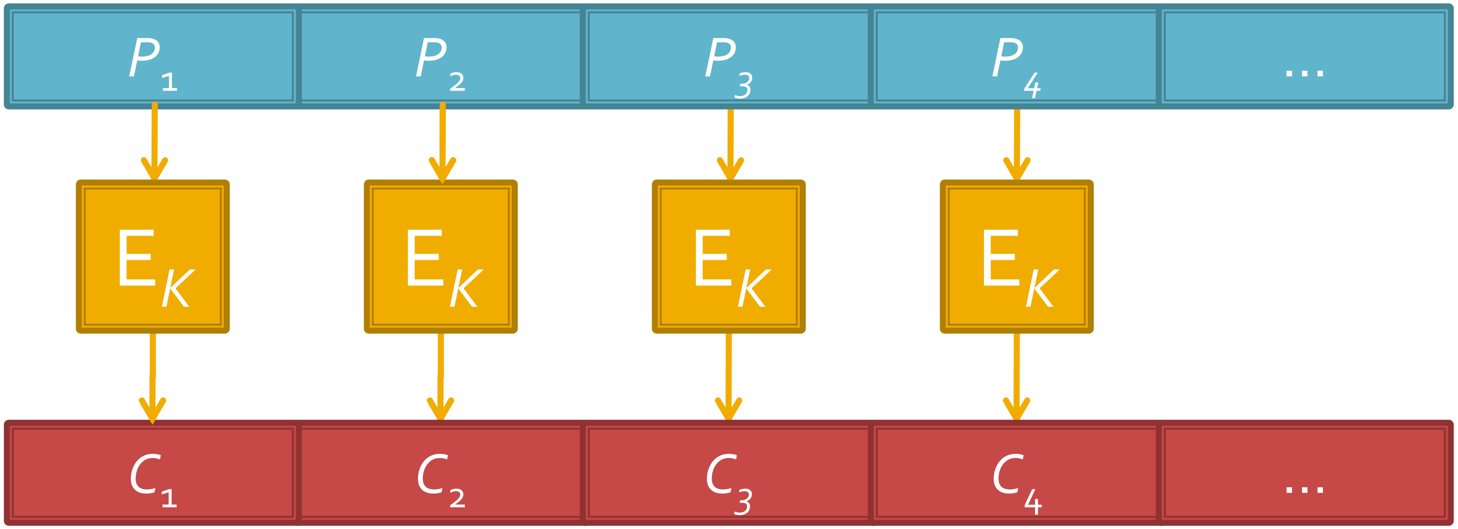
Block Ciphers (review)





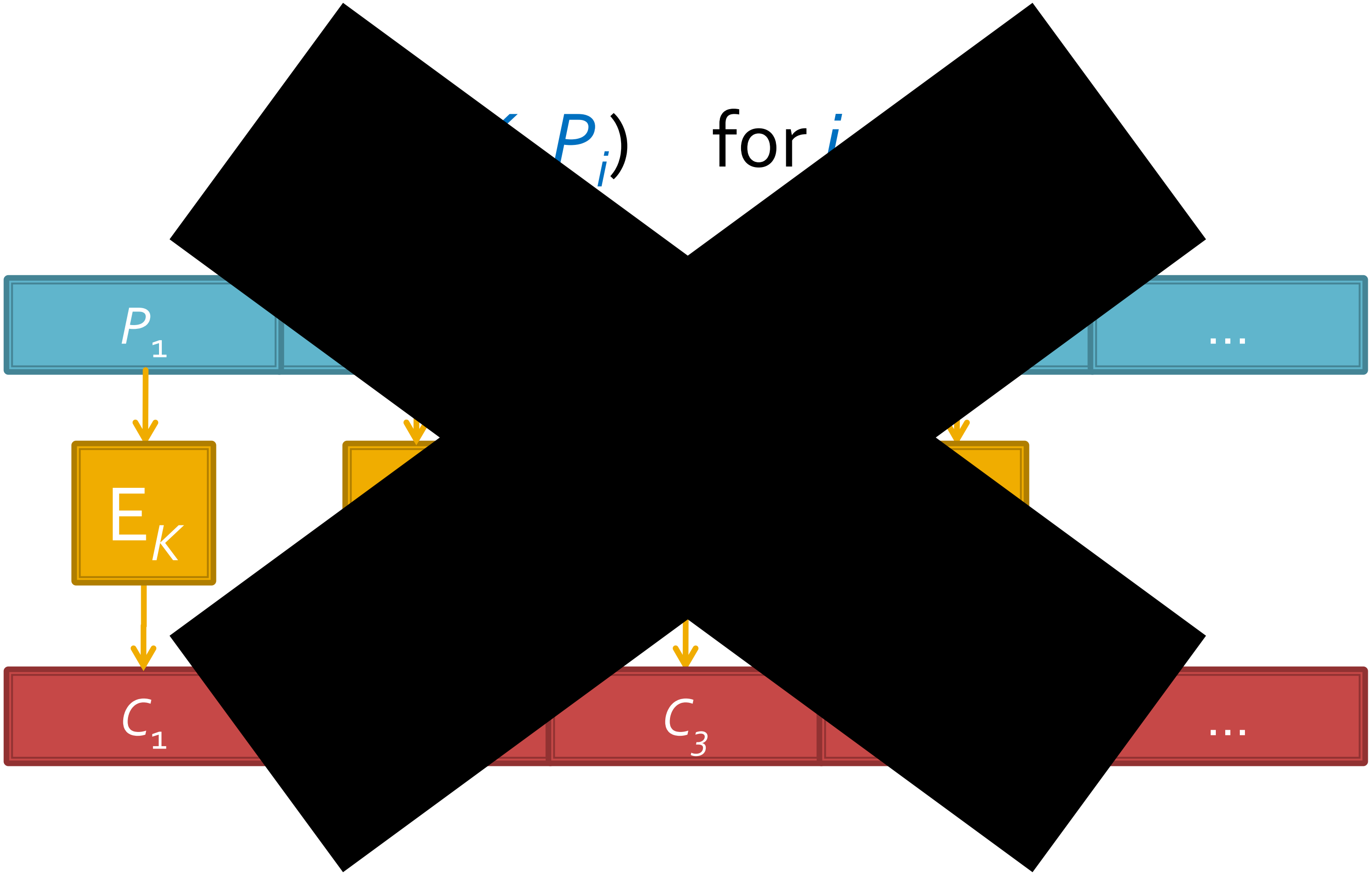
ECB – Electronic Codebook Mode

$$C_i := \mathrm{E}(K, P_i) \quad \text{for } i = 1, \ldots, n$$



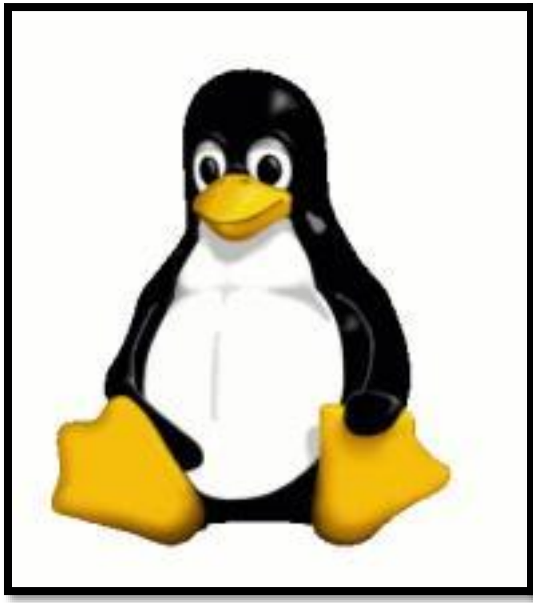
ECB – Electronic Codebook Mode

$P_i)$ for i

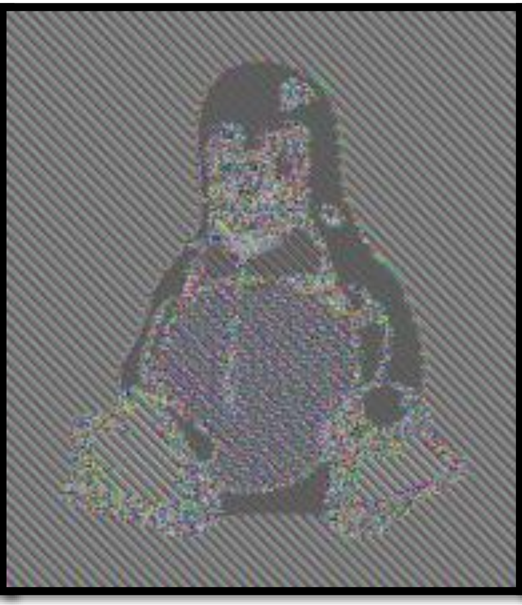


Why not ECB?

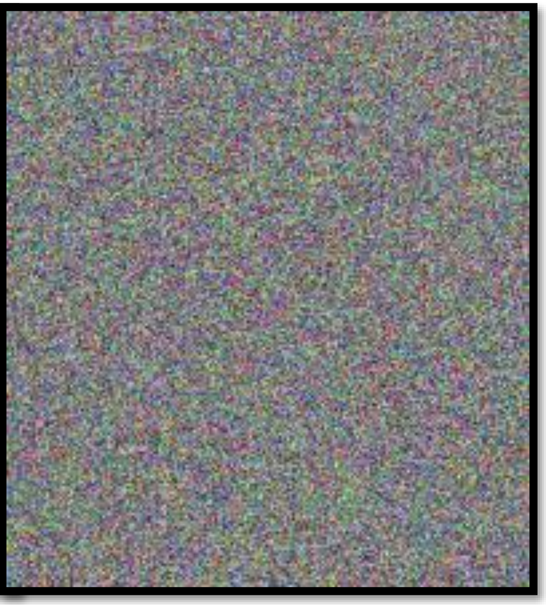
- The cipher text of an identical block is always identical… consider a bitmap image…



(plaintext)



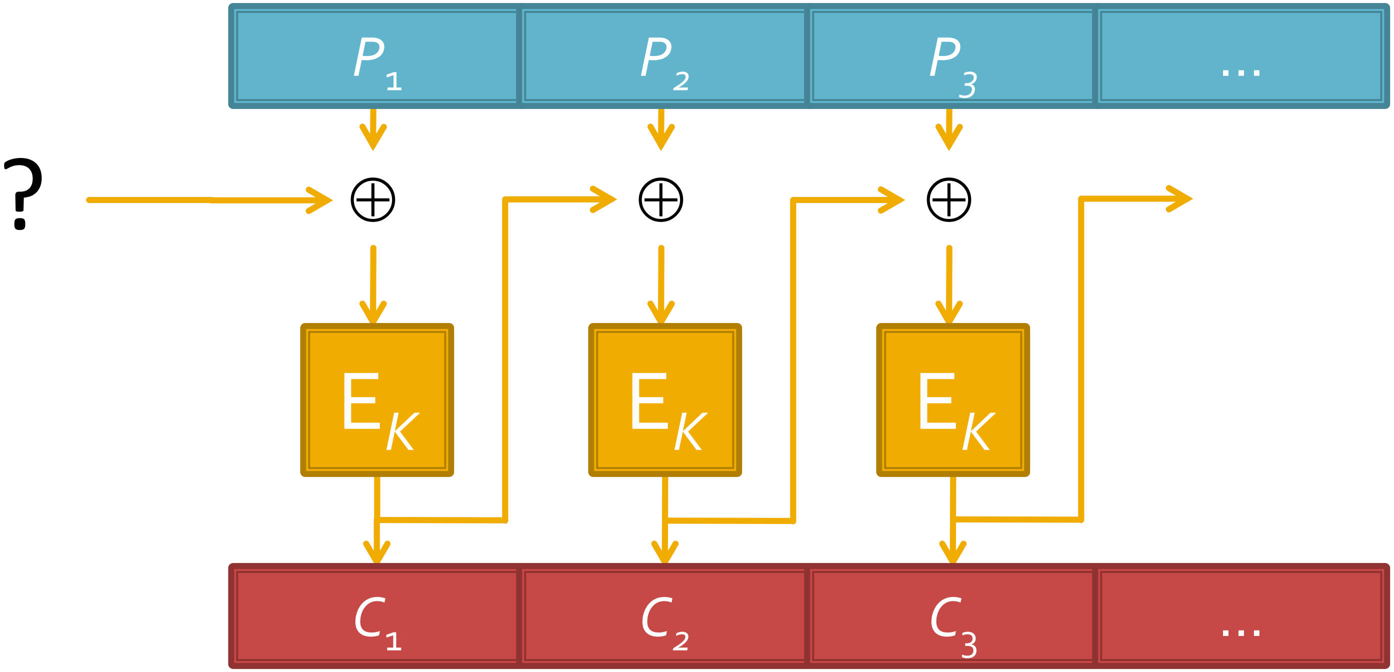
(ECB mode)



(CBC mode)

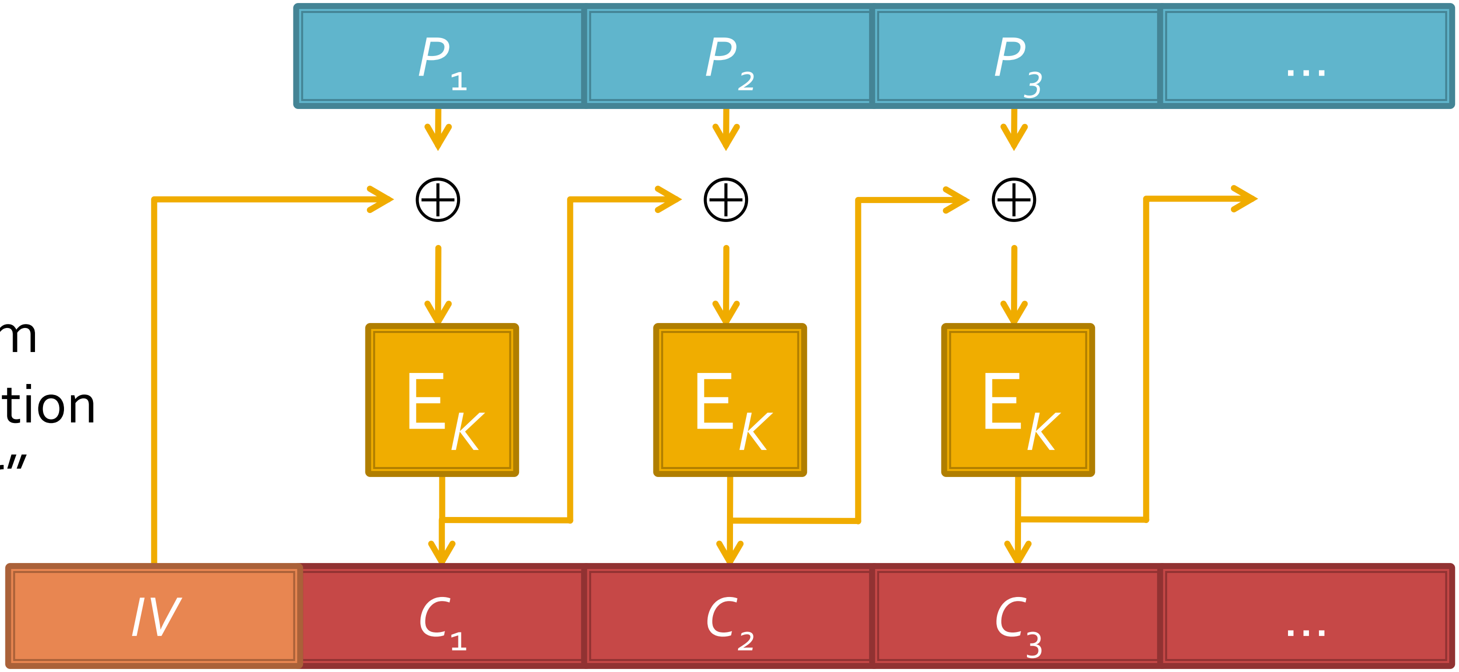
CBC: Cipher-Block Chaining Mode

$C_i := \mathrm{E}(K, P_i \oplus C_{i-1})$ for $i = 1, \ldots, n$



CBC: Cipher-Block Chaining Mode

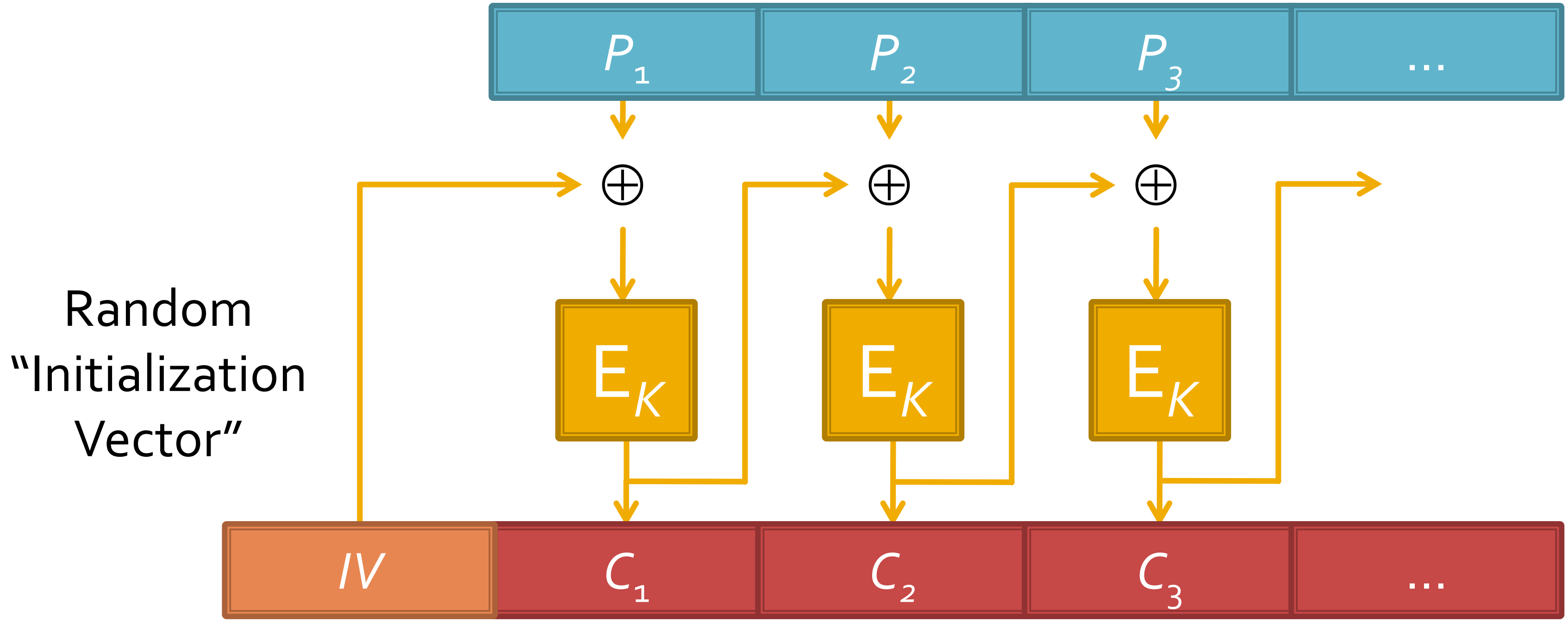
$C_i := \mathrm{E}(K, P_i \oplus C_{i-1})$ for $i = 1, \ldots, n$



Random
"Initialization
Vector"

CBC: Cipher-Block Chaining Mode

$C_i := E(K, P_i \oplus C_{i-1})$ for $i = 1, \ldots, n$



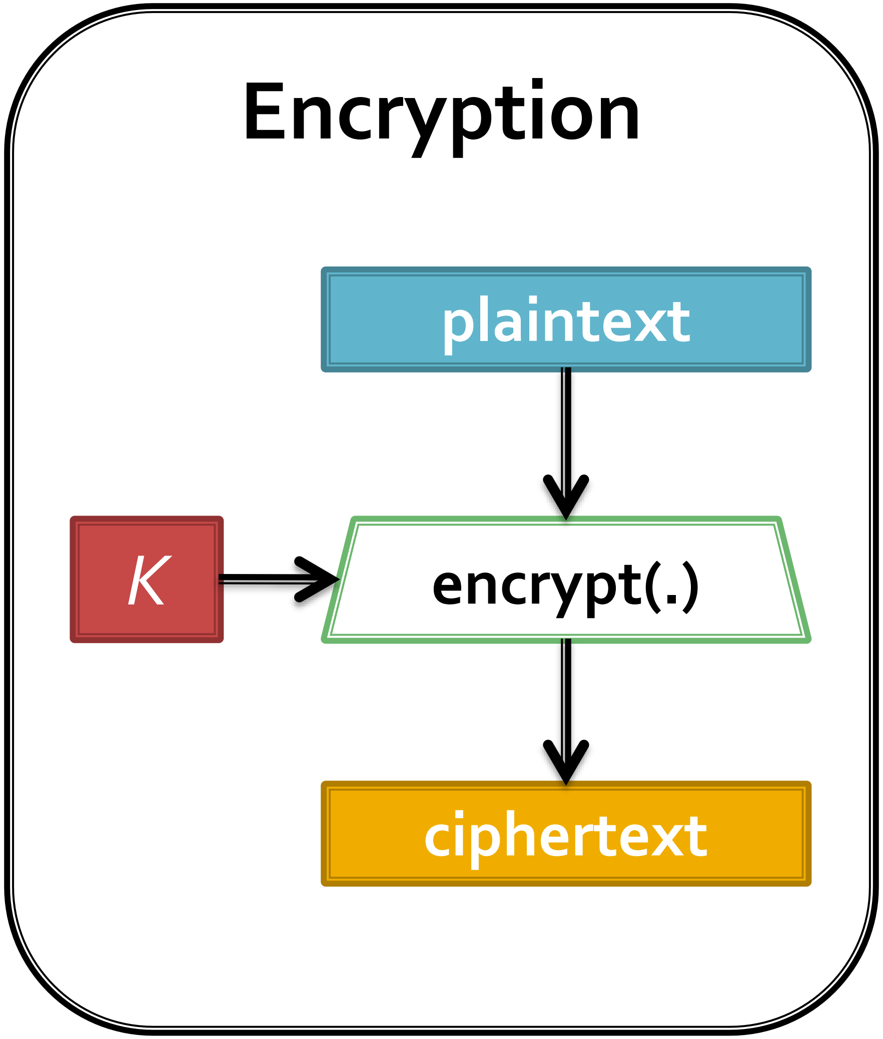
DO NOT REUSE INITIALIZATION VECTORS!!

CTR: Counter Mode

$$K_i := \mathrm{E}(K, \mathit{Nonce} \,\|\, i) \quad \text{for } i = 1, \ldots, n$$
$$C_i := P_i \oplus K_i$$

- Stream cipher construction
- Plaintext never passes through *E*
- Don’t need to pad the message
- Allows parallelization and seeking
- Never reuse same *K*+*Nonce*

Symmetric Key Encryption



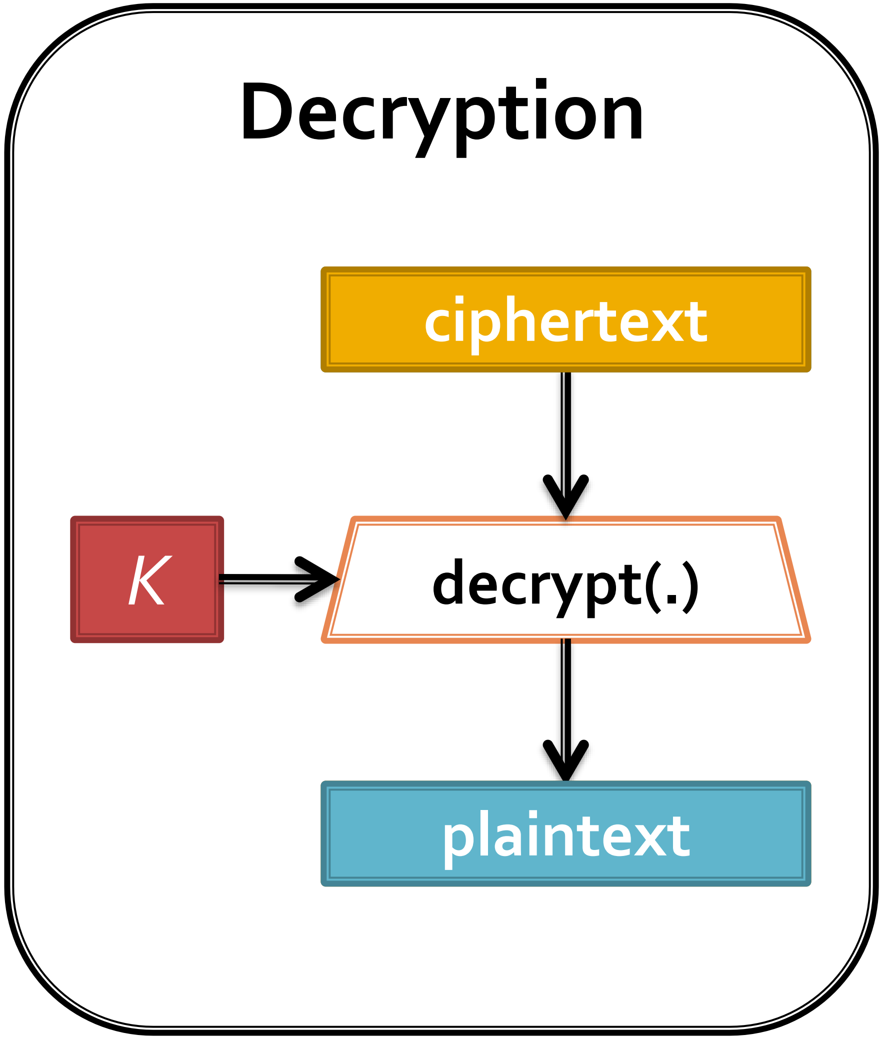
Encryption

plaintext

K

encrypt(.)

ciphertext



Decryption

ciphertext

K

decrypt(.)

plaintext

Public Key Cryptography

- Symmetric key cryptographic is great… but has the fundamental problem that every send-receiver pair must share a secret key…
- How do we allow the sender and receiver to use different keys for encryption and decryption?
- Also known as “Asymmetric Encryption”

Diffie-Hellman Key Exchange

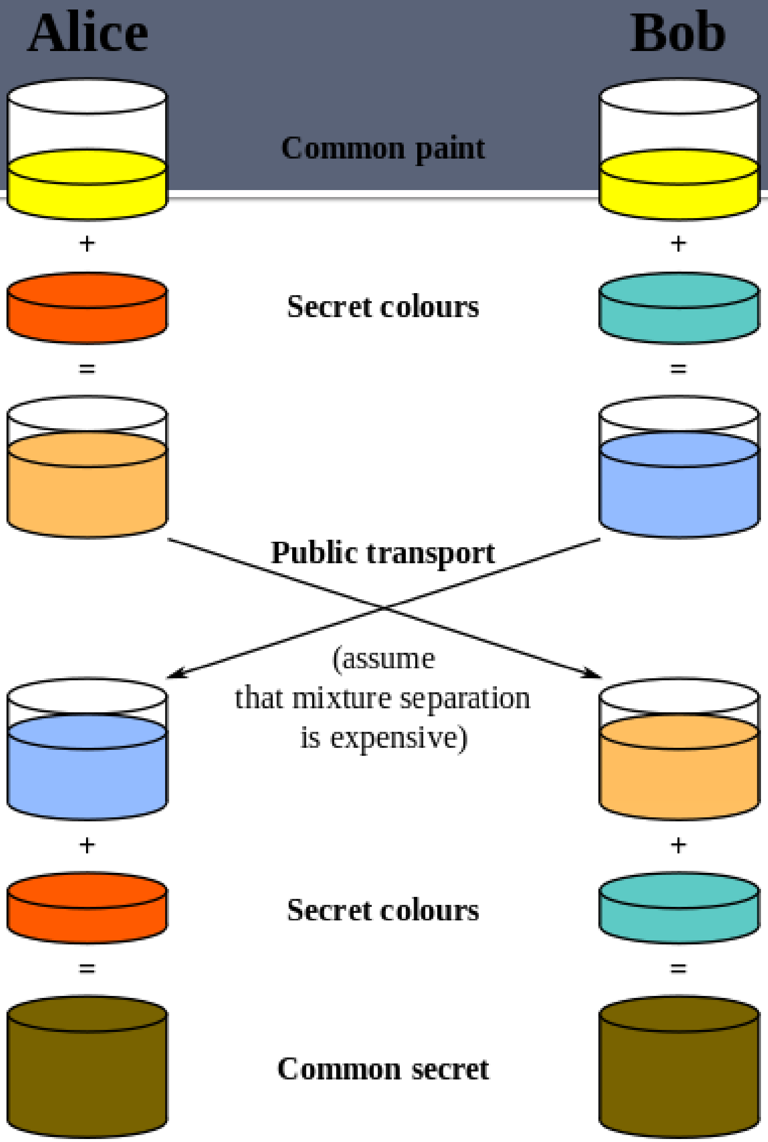
- How do we share our symmetric key in front of an eavesdropping adversary?
- “Key Exchange” developed by Whitfield Diffie and Martin Hellman in 1976
- Based on *Discrete Log Problem* which we believe is difficult (“the assumption”)

Diffie-Hellman Key Exchange

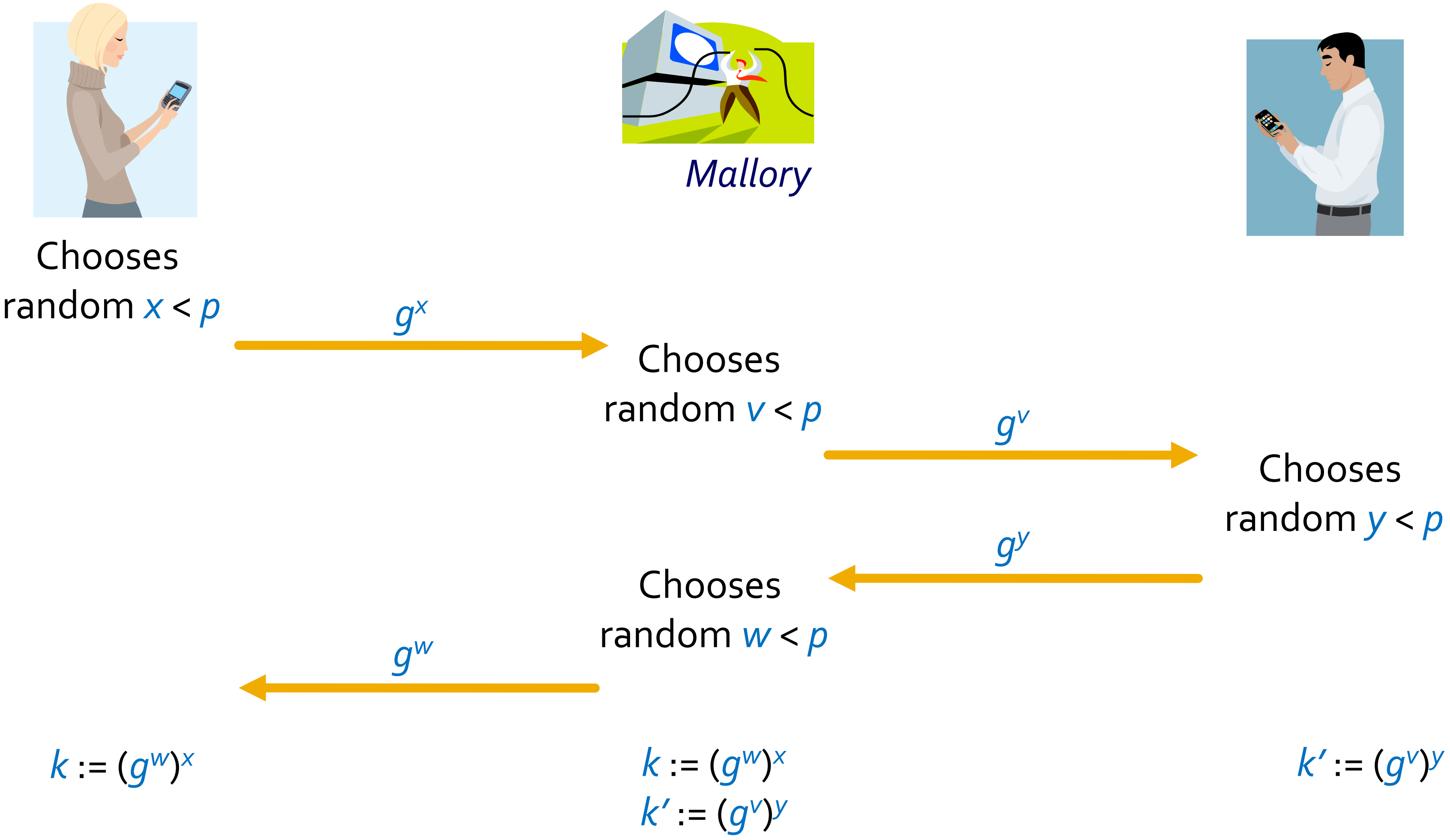
1. Alice generates and shares ***g*** with Bob
2. Alice and Bob each generate a secret number, which we denote ***a*** and ***b***
3. Alice generates g^a and sends it to Bob
4. Bob generates g^b and sends it to Alice
5. Alice calculates $(g^b)^a$ and Bob calculates $(g^a)^b$
6. Alice and Bob have $(g^b)^a = g^{ab} = g^{ba} = (g^a)^b$

Some Diffie-Hellman Details

1. D-H works in any finite cyclic group. Assume *G* is predetermined and we are selecting a generator g Î G
2. We almost always just use Z_p^* (multiplicative group of integers modulo p)
3. We share a primitive root (***g***) and an odd prime (***p***) and perform all operations mod ***p***.


Alice
Bob
Common paint
+
+
Secret colours
=
=
Public transport
(assume
that mixture separation
is expensive)
+
+
Secret colours
=
=
Common secret

Attacking Diffie-Hellman (MITM)



Mallory

Chooses random $x < p$

g^x

Chooses random $v < p$

g^v

Chooses random $y < p$

g^y

Chooses random $w < p$

g^w

$k := (g^w)^x$

$k := (g^w)^x$
$k' := (g^v)^y$

$k' := (g^v)^y$

Summary of Goals

✓ Confidentiality

✓ Integrity

✗ Authentication

RSA Public Key Encryption

RSA Encryption

p, q	large random primes
$n := pq$	modulus
$t := (p-1)(q-1)$	ensures $x^t = 1 \pmod{n}$
$e :=$ [small prime value]	public exponent
$d := e^{-1} \bmod t$	private exponent

Public key: (n, e)

Private key: (p, q, t, d)

RSA Encryption

1. Public Key: (n, e)
2. Private Key: (p, q, t, d)

3. Encryption: $c := m^e \bmod n$
4. Decryption: $m := c^d \bmod n$

5. $(m^e)^d = m^{ed} = m^{kt+1} = (m^t)^k m = 1^k m = m \pmod{n}$

Encryption with RSA

1. Public Key Encryption is much slower than symmetric key encryption
2. Publish public key to the world, keep private key secret
3. Negotiate a symmetric key over public key encryption and utilize the symmetric key for encrypting any actual data going forward

Other Public Key Algorithms

- Other public key algorithms do exist

- ElGamal (digital signature scheme based on DL)
- DSA (Digital Signature Algorithm)
- Elliptic Curve DSA (ECDSA)

- ECDSA is quickly gaining popularity

Establishing Trust

- **How do Alice and Bob share public keys?**

- Web of Trust (e.g. PGP)
- Trust on First Use (TOFU) (e.g. SSH)
- Public Key Infrastructure (PKI) (e.g. SSL)

What is PKI?

- Organizations we trust (often known as "Certificate Authorities") generate certificates to tie a public key to an organization

- We trust that we're talking to the correct organization if we can verify their public key with a trusted authority

SSL/TLS Certificates

Subject: C=US/O=Google Inc/CN=www.google.com
Issuer: C=US/O=Google Inc/CN=Google Internet Authority
Serial Number: 01:b1:04:17:be:22:48:b4:8e:1e:8b:a0:73:c9:ac:83
Expiration Period: Jul 12 2010 - Jul 19 2012
Public Key Algorithm: rsaEncryption
Public Key: 43:1d:53:2e:09:ef:dc:50:54:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:39:23:46

Signature Algorithm: sha1WithRSAEncryption

Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:1e:5d:b5

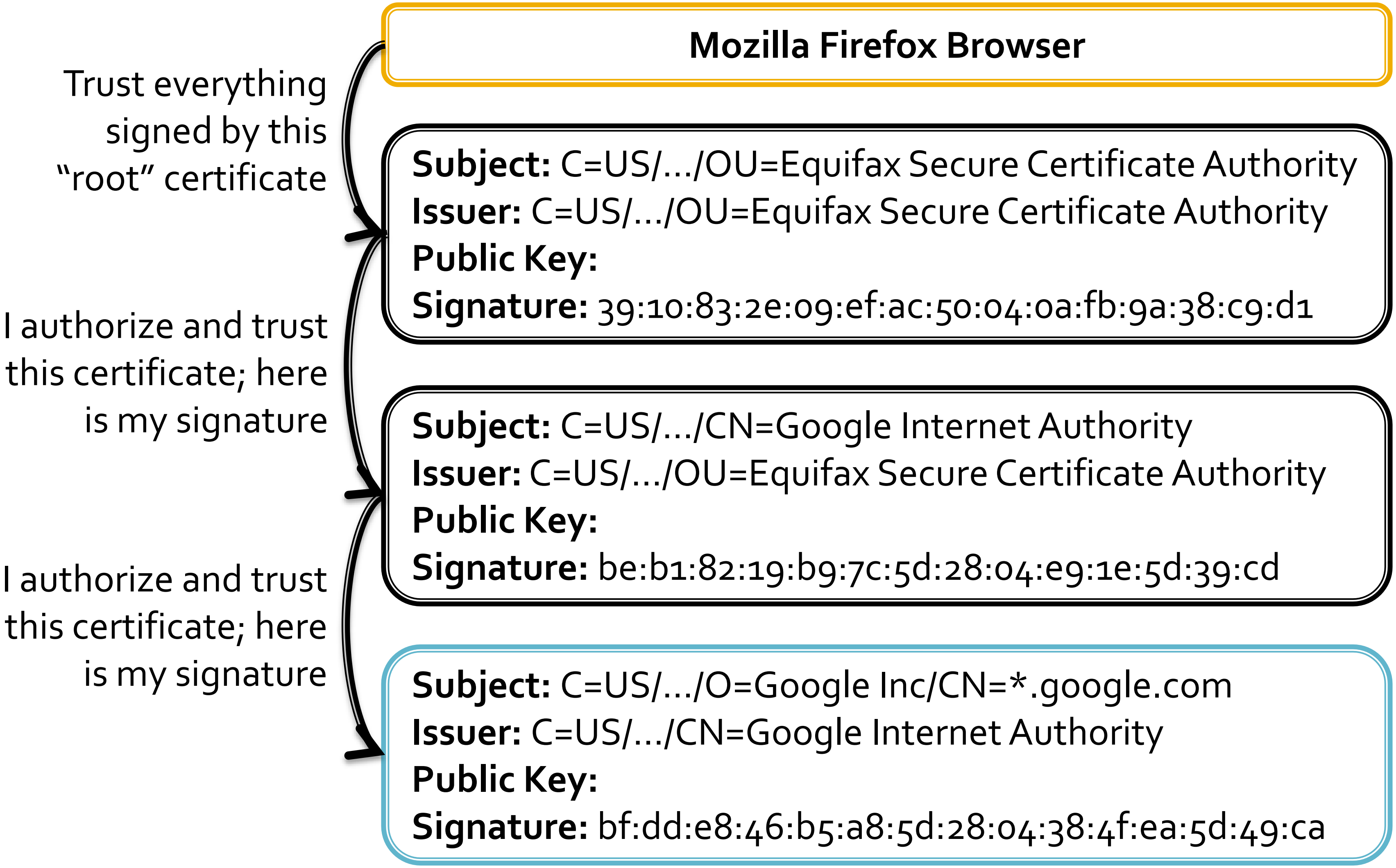
Signatures on Certificates

- Utilize both public key cryptography and cryptographic hash functions

- Oftentimes see a signature algorithm such as **`sha1WithRSAEncryption`**

- $\mathbf{\texttt{Encrypt}_{\texttt{PrivateKey}}\texttt{(SHA-1(certificate))}}$

Certificate Chains



Mozilla Firefox Browser

Trust everything signed by this "root" certificate

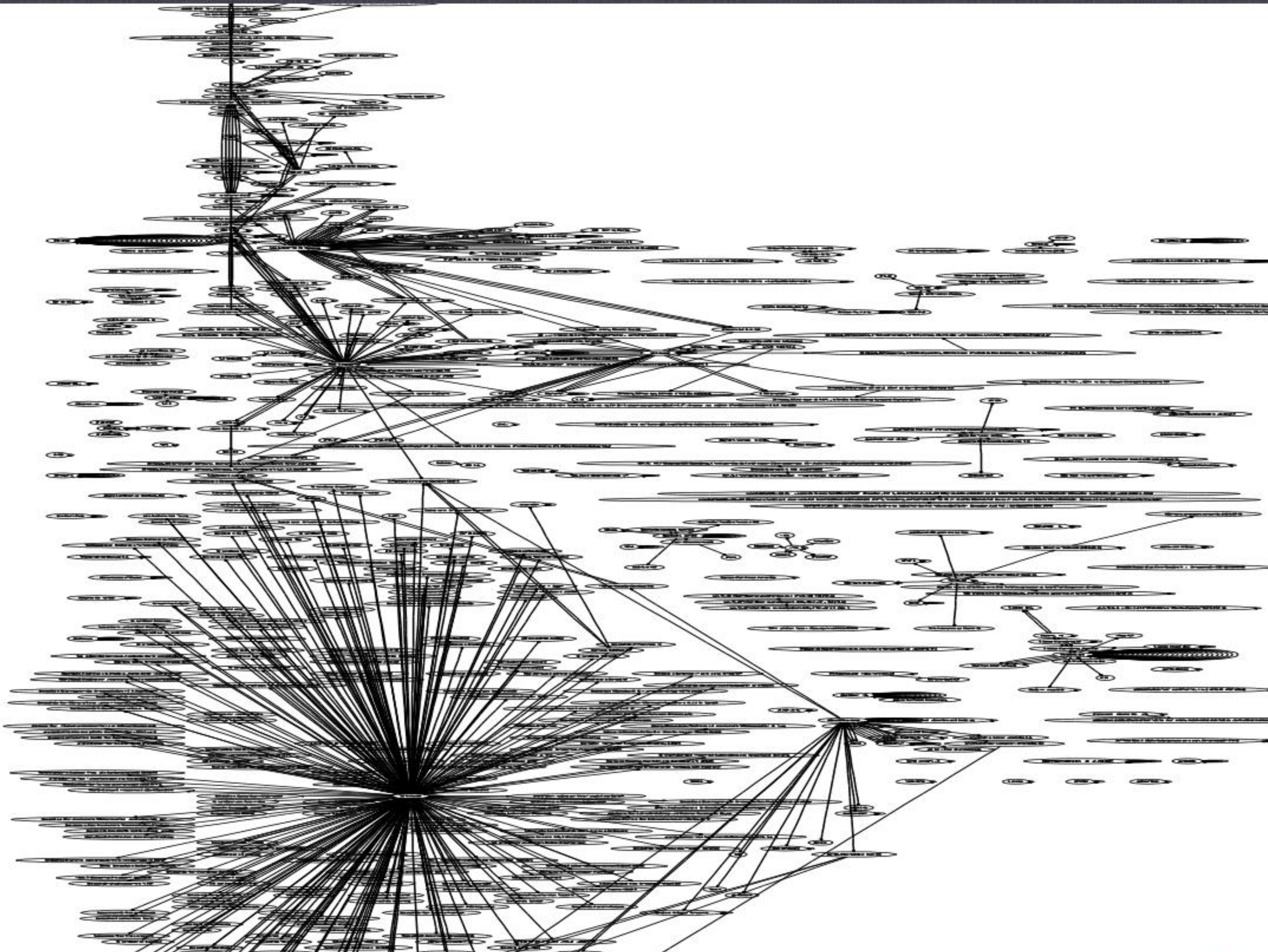
Subject: C=US/…/OU=Equifax Secure Certificate Authority
Issuer: C=US/…/OU=Equifax Secure Certificate Authority
Public Key:
Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:38:c9:d1

I authorize and trust this certificate; here is my signature

Subject: C=US/…/CN=Google Internet Authority
Issuer: C=US/…/OU=Equifax Secure Certificate Authority
Public Key:
Signature: be:b1:82:19:b9:7c:5d:28:04:e9:1e:5d:39:cd

I authorize and trust this certificate; here is my signature

Subject: C=US/…/O=Google Inc/CN=*.google.com
Issuer: C=US/…/CN=Google Internet Authority
Public Key:
Signature: bf:dd:e8:46:b5:a8:5d:28:04:38:4f:ea:5d:49:ca



Some Practical Advice

- **HMAC:** *HMAC-SHA256*
- **Block Cipher:** *AES-256*
- **Randomness:** OS Cryptographic Pseudo Random Number Generator (CPRNG)
- **Public Key Encryption:** *RSA* or *ECDSA*
- **Implementation:** *OpenSSL*

Related Research Problems

- *Cryptanalysis:* Ongoing work to break crypto functions… rapid progress on hash collisions
- *Cryptographic function design:* We badly need better hash functions… NIST competition now to replace SHA
- *Attacks:* Only beginning to understand implications of MD5 breaks – likely enables many major attacks

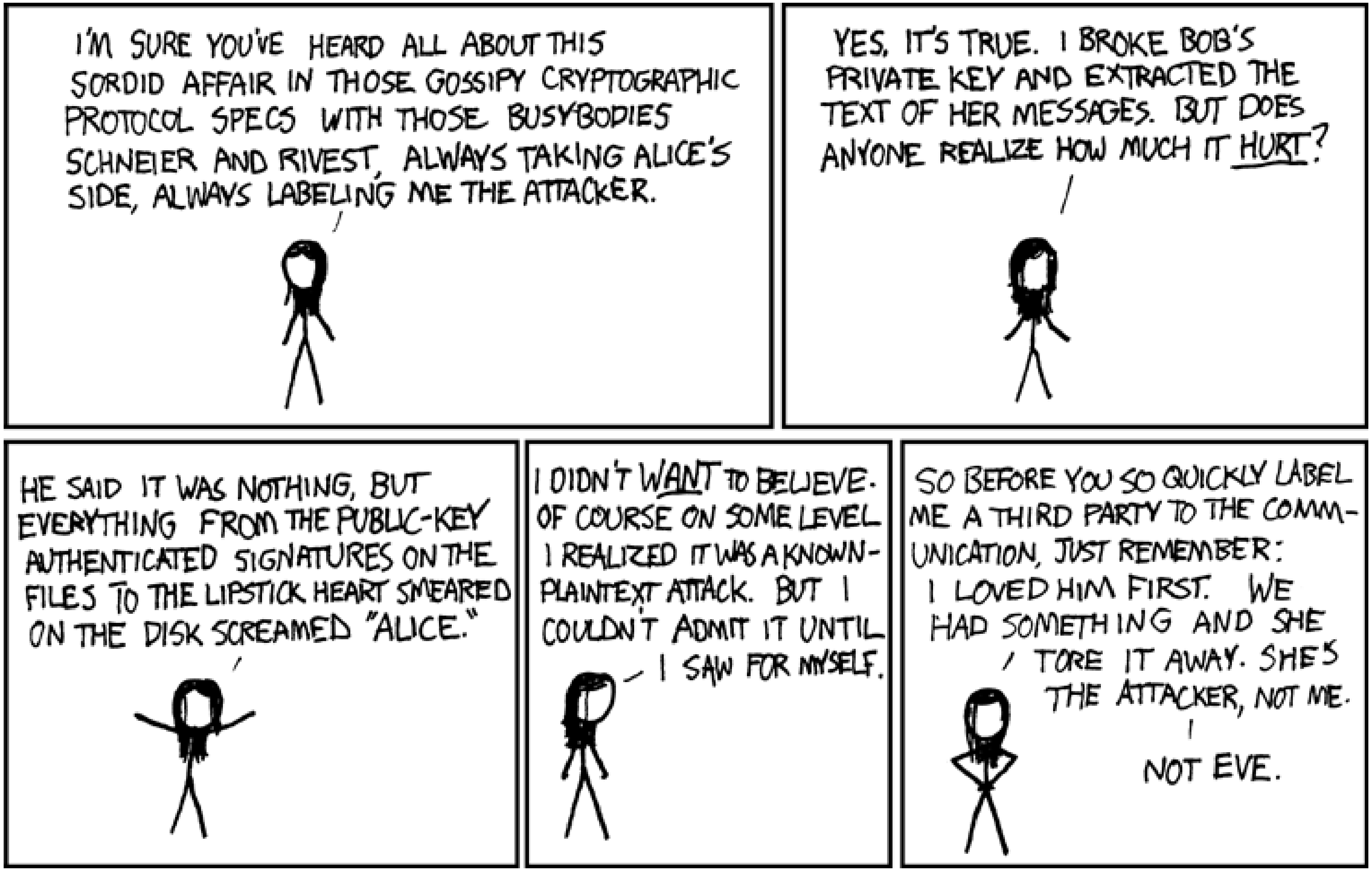
Don’t Roll Your Own!!

CATS: ALL YOUR BASE ARE BELONG TO US.

SECRIT: Security Reading Group

- We read a recent security paper and discuss it over lunch each week
- Tuesdays from 12:30 to 1:30 PM
- (one read paper) == (one free lunch)
- https://wiki.eecs.umich.edu/secrit/

Tuesday: Alex's Introduction


I'M SURE YOU'VE HEARD ALL ABOUT THIS SORDID AFFAIR IN THOSE GOSSIPY CRYPTOGRAPHIC PROTOCOL SPECS WITH THOSE BUSYBODIES SCHNEIER AND RIVEST, ALWAYS TAKING ALICE'S SIDE, ALWAYS LABELING ME THE ATTACKER.
YES, IT'S TRUE. I BROKE BOB'S PRIVATE KEY AND EXTRACTED THE TEXT OF HER MESSAGES. BUT DOES ANYONE REALIZE HOW MUCH IT HURT?
HE SAID IT WAS NOTHING, BUT EVERYTHING FROM THE PUBLIC-KEY AUTHENTICATED SIGNATURES ON THE FILES TO THE LIPSTICK HEART SMEARED ON THE DISK SCREAMED "ALICE."
I DIDN'T WANT TO BELIEVE. OF COURSE ON SOME LEVEL I REALIZED IT WAS A KNOWN-PLAINTEXT ATTACK. BUT I COULDN'T ADMIT IT UNTIL I SAW FOR MYSELF.
SO BEFORE YOU SO QUICKLY LABEL ME A THIRD PARTY TO THE COMMUNICATION, JUST REMEMBER: I LOVED HIM FIRST. WE HAD SOMETHING AND SHE TORE IT AWAY. SHE'S THE ATTACKER, NOT ME.
NOT EVE.