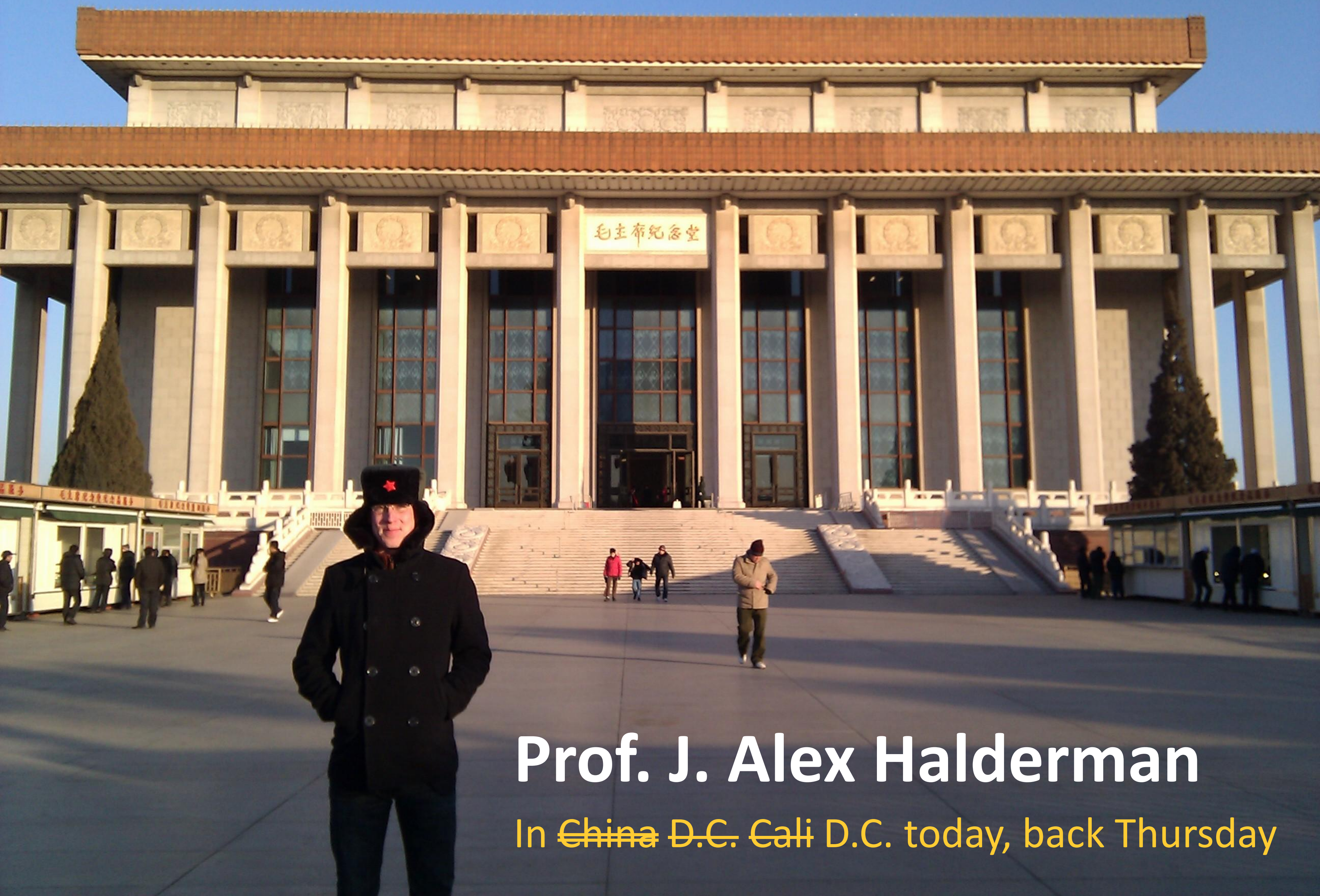
Essential Cryptography Part II

EECS588 Computer and Network Security

January 15, 2013

The Itinerant Professor



Prof. J. Alex Halderman

In ~~China~~ ~~D.C.~~ ~~Cali~~ D.C. today, back Thursday

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 SSL 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

Communication

Course Web Site
https://www.eecs.umich.edu/courses/eecs588/
announcements, schedule, readings

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

Goals of Cryptography

- **Confidentiality:** only the intended recipient should be able to decrypt the cipher text
- **Integrity:** the recipient should be able to detect whether a message has been altered
- **Authentication:** how do we verify the identity of the sender?
- **(Non-)repudiation:** the sender should not be able to deny sending the message

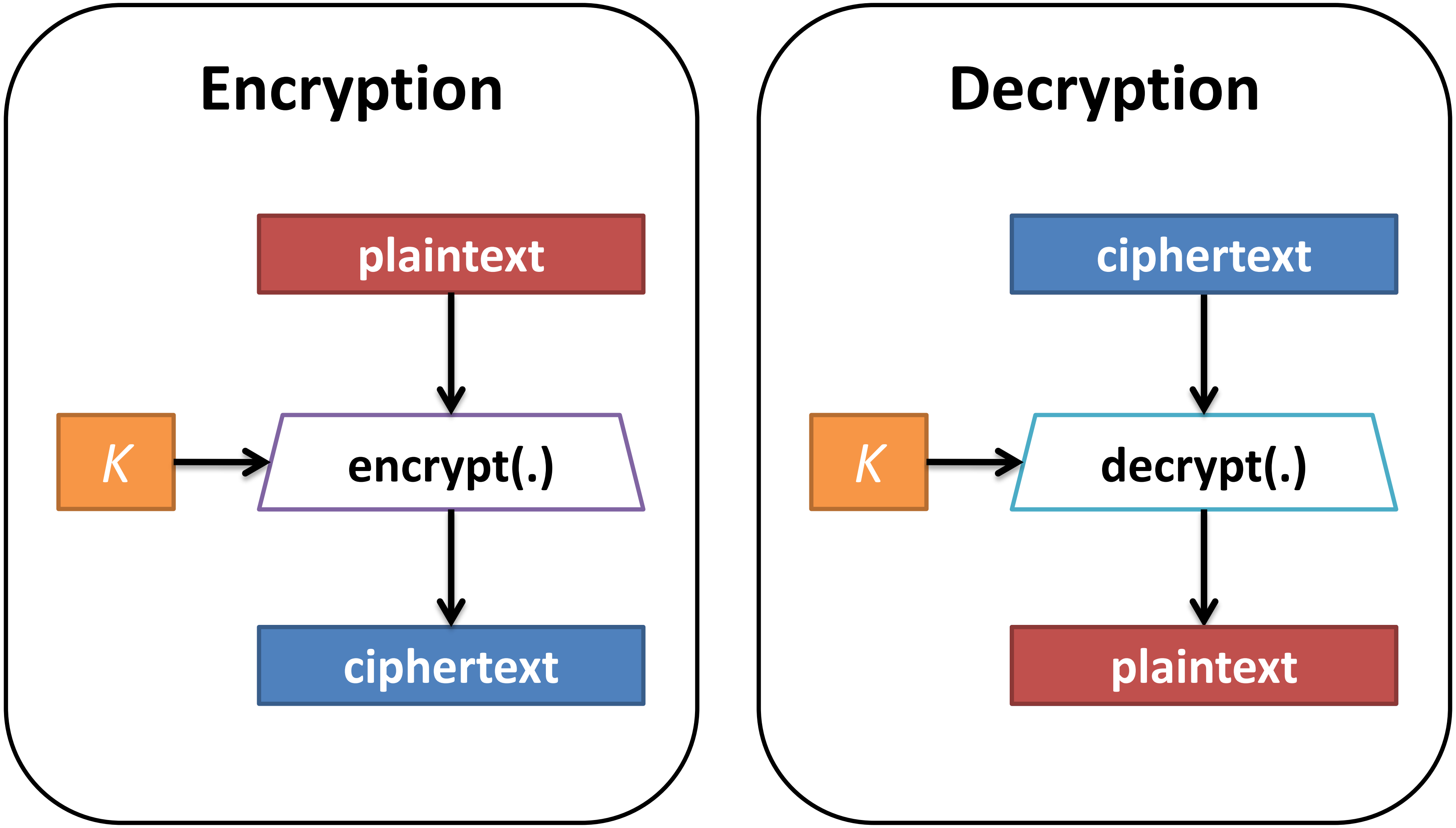
Don’t Roll Your Own!!

Common Block Ciphers

- Common:
  - **AES (Advanced Encryption Standard)**
  - RC5
  - 3DES (“triple DES”)
  - Blowfish

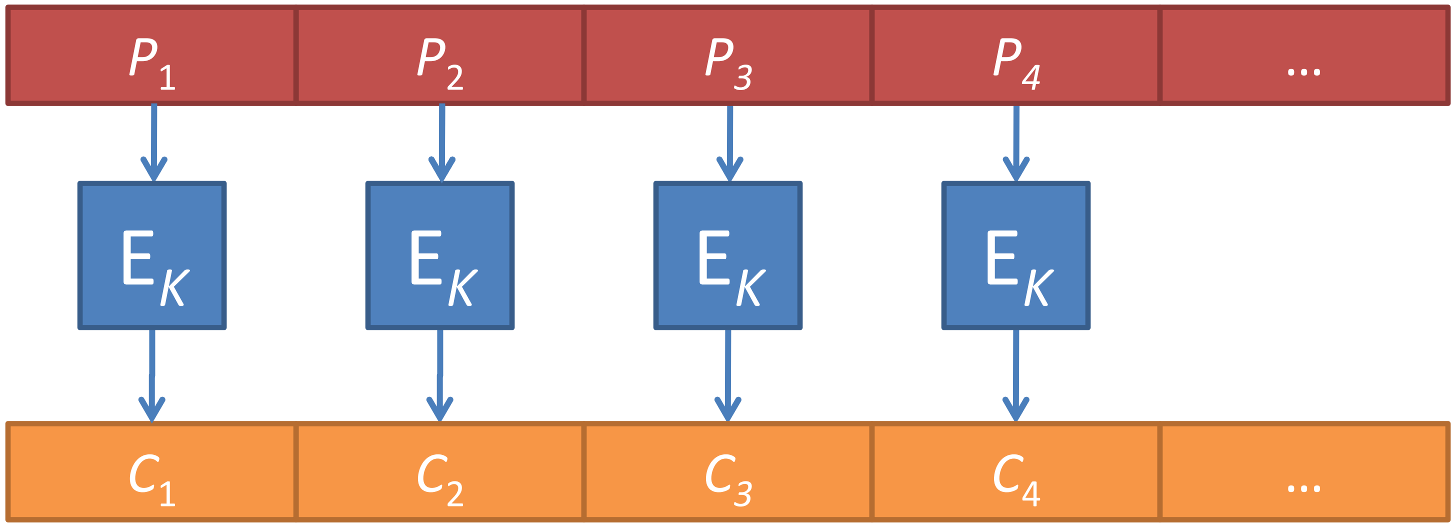
- Broken:
  - DES (don’t use!)

Block Ciphers (review)

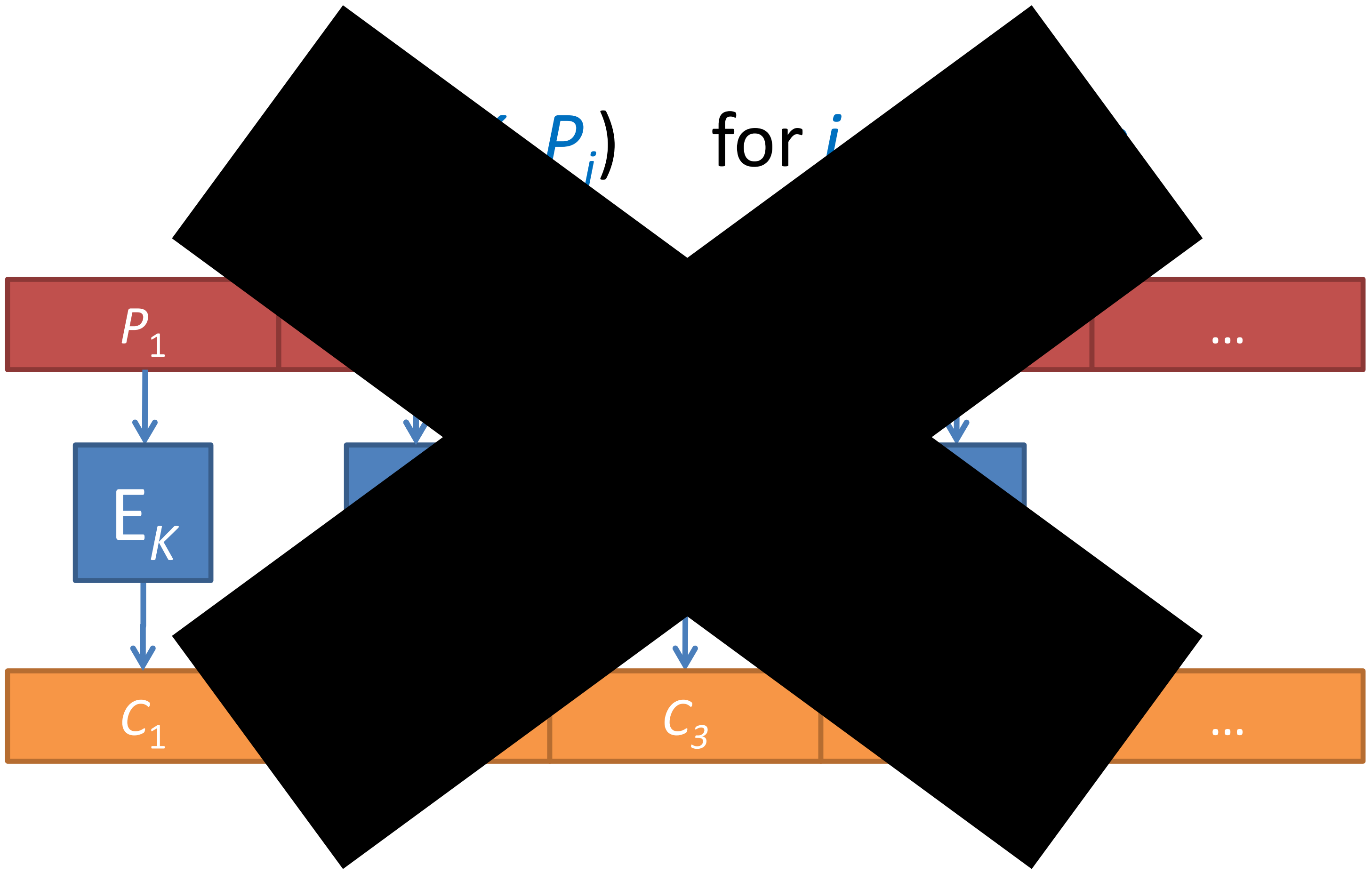


ECB – Electronic Codebook Mode

C_i := E(K, P_i)   for i = 1, …, n

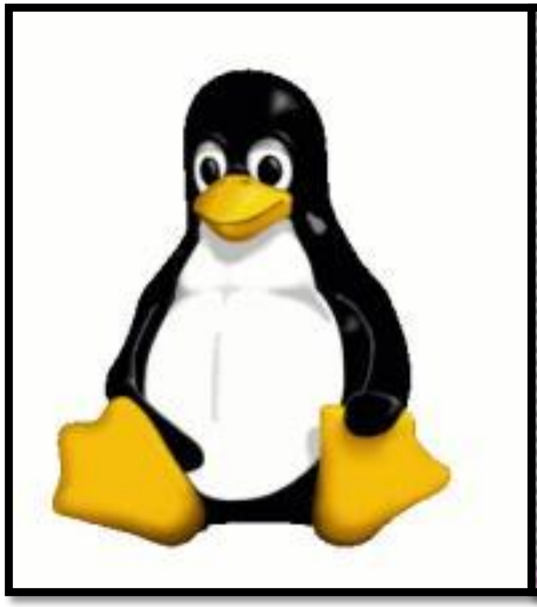


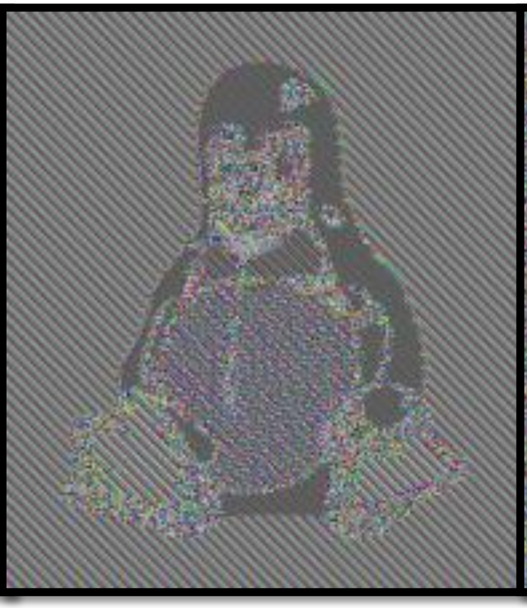
ECB – Electronic Codebook Mode

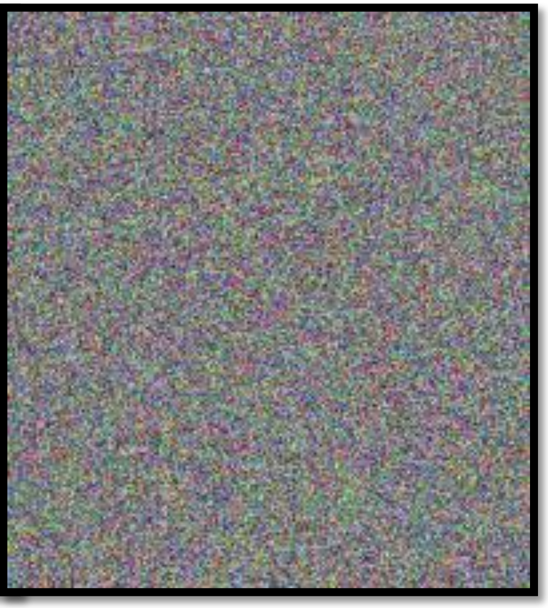


Why not ECB?

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

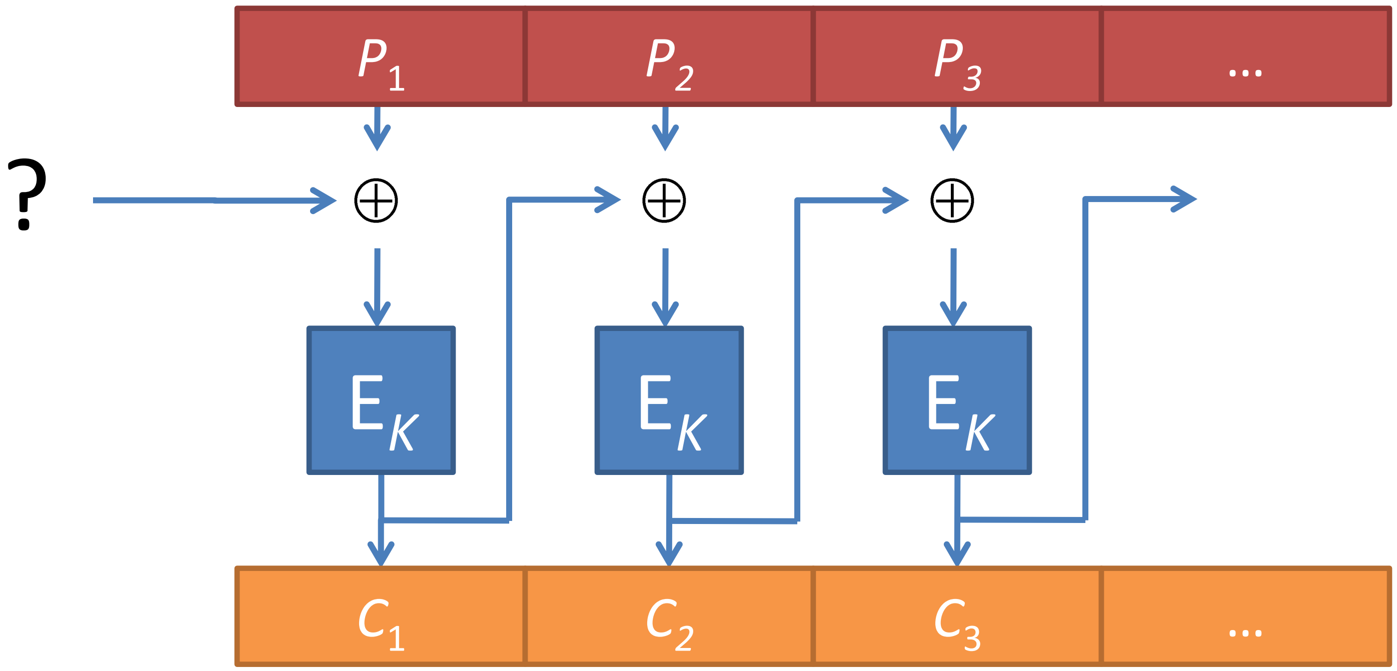

(plaintext)


(ECB mode)


(CBC mode)

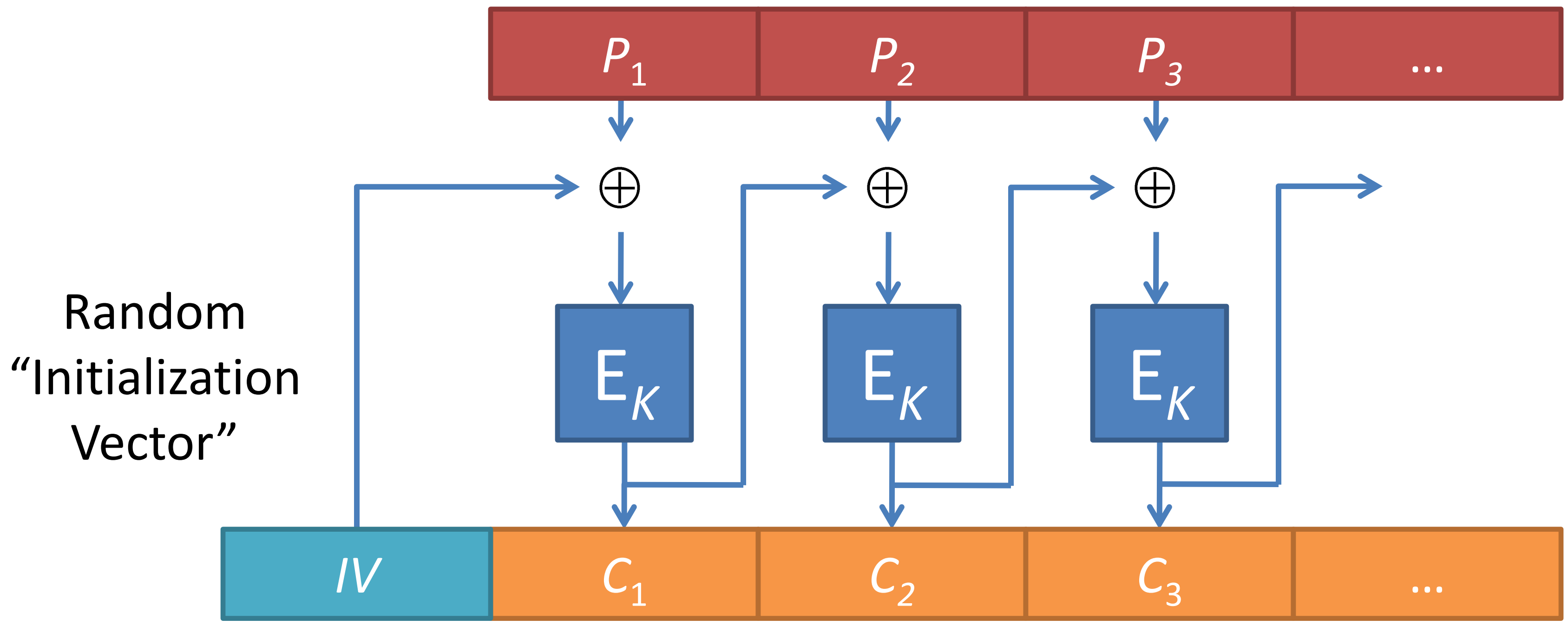
CBC: Cipher-Block Chaining Mode

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



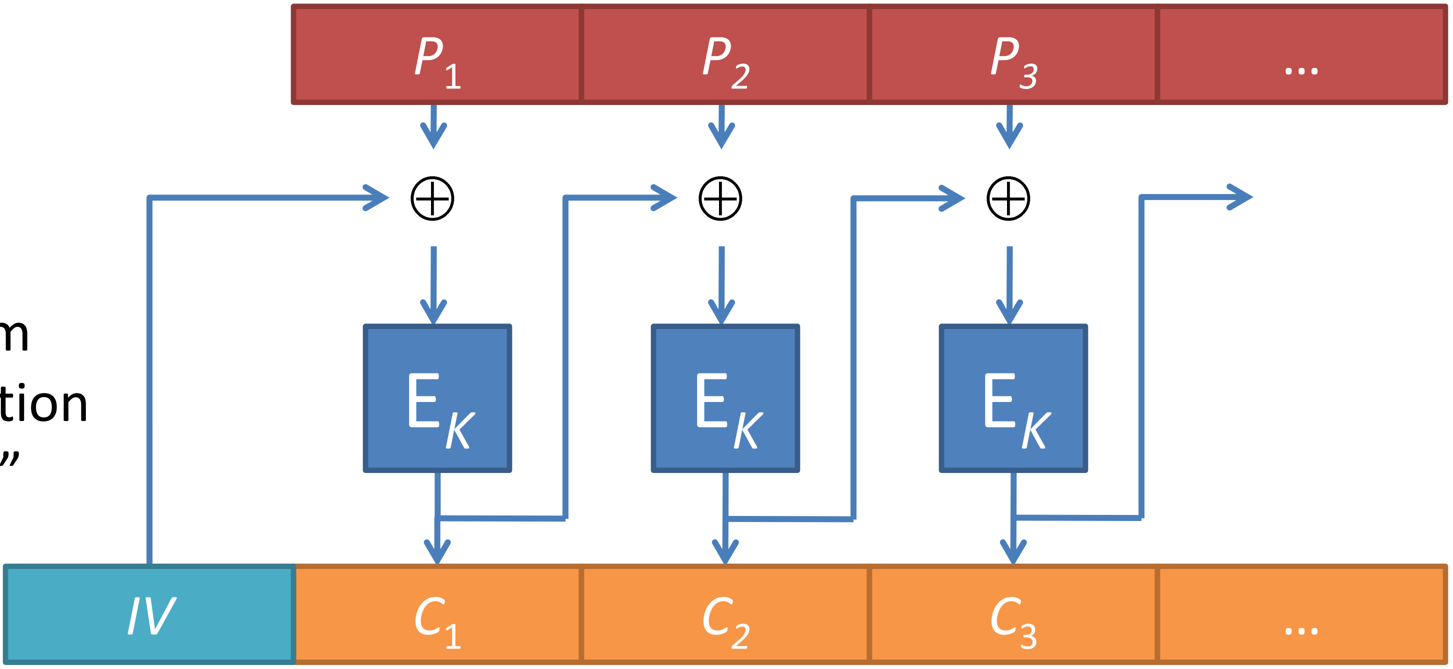
CBC: Cipher-Block Chaining Mode

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



CBC: Cipher-Block Chaining Mode

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



Random "Initialization Vector"

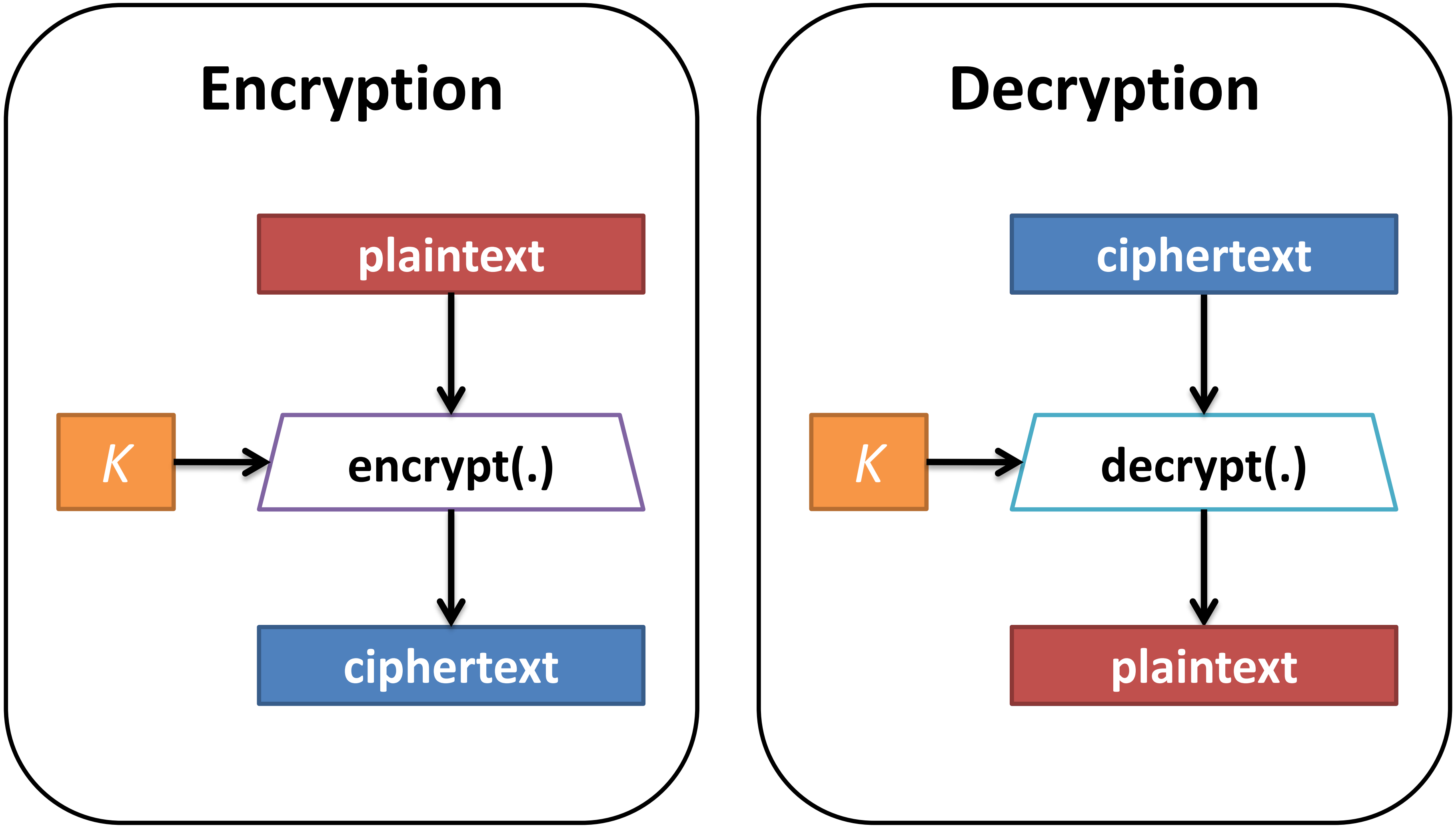
DO NOT REUSE INITIALIZATION VECTORS!!

CTR: Counter Mode

K_i := E(K, $Nonce$ || i)   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
- <u>Never</u> reuse same *K+Nonce*

Symmetric Key Encryption



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

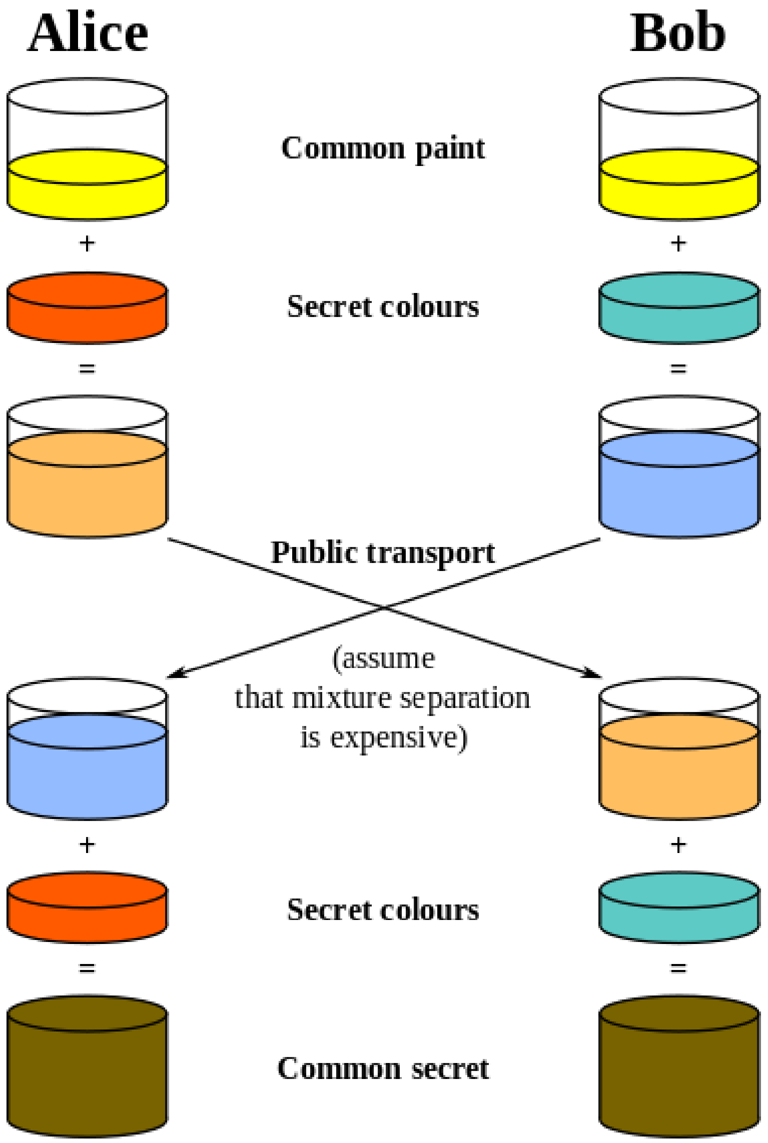
- How do we share our symmetric key in front of an eavesdropping adversary?
- “Key Exchange” developed by Whitfield Diffie and Martin Hellman in 1976
- Based on *Discrete Log Problem* which we believe is difficult (“the assumption”)

Diffie-Hellman Key Exchange

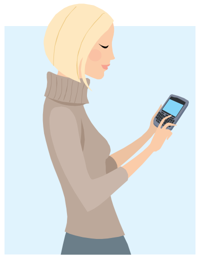
1. Alice generates and shares $\boldsymbol{g}$ with Bob
2. Alice and Bob each generate a secret number, which we denote $\boldsymbol{a}$ and $\boldsymbol{b}$
3. Alice generates $\boldsymbol{g^a}$ and sends it to Bob
4. Bob generates $\boldsymbol{g^b}$ and sends it to Alice
5. Alice calculates $(g^b)^a$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

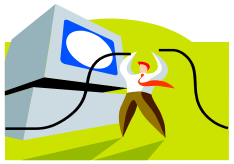
D-H for People Who Know Math

1. D-H works in any finite cyclic group. Assume *G* is predetermined and we are selecting a generator $g \in G$
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***.

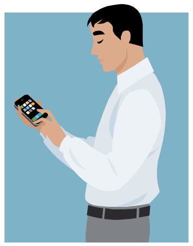

Alice
Bob
Common paint
+
Secret colours
=
Public transport
(assume
that mixture separation
is expensive)
+
Secret colours
=
Common secret

Attacking Diffie-Hellman (MITM)





Mallory



Chooses random $x < p$

$\xrightarrow{g^x}$

Chooses random $v < p$

$\xrightarrow{g^v}$

Chooses random $y < p$

$\xleftarrow{g^y}$

Chooses random $w < p$

$\xleftarrow{g^w}$

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

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

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

Summary of Goals

✓ Confidentiality

✓ Integrity

✗ Authentication

RSA Public Key Encryption

RSA Encryption

p, q	large random primes
$n := pq$	modulus
$t := (p\text{-}1)(q\text{-}1)$	ensures $x^t = 1 \pmod{n}$
$e :=$ [small odd 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

RSA for Encryption

- Publish: (n, e), Store secretly: d
- Encryption of m

  Choose random k same size as n

  $c := k^e \bmod n$

  Send c, encrypt m with AES using k
- Decryption

  $k := c^d \bmod n$; decrypt m with AES using k

RSA for Signatures

- Publish: (n, e), Store secretly: d
- Signing m

  Seed a CPRNG with m and calculate pseudorandom string s same size as n

  $\sigma := s^d \bmod n$
- Verifying a signature on m

  Recalculate s from m

  Check $s = \sigma^e \bmod n$

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

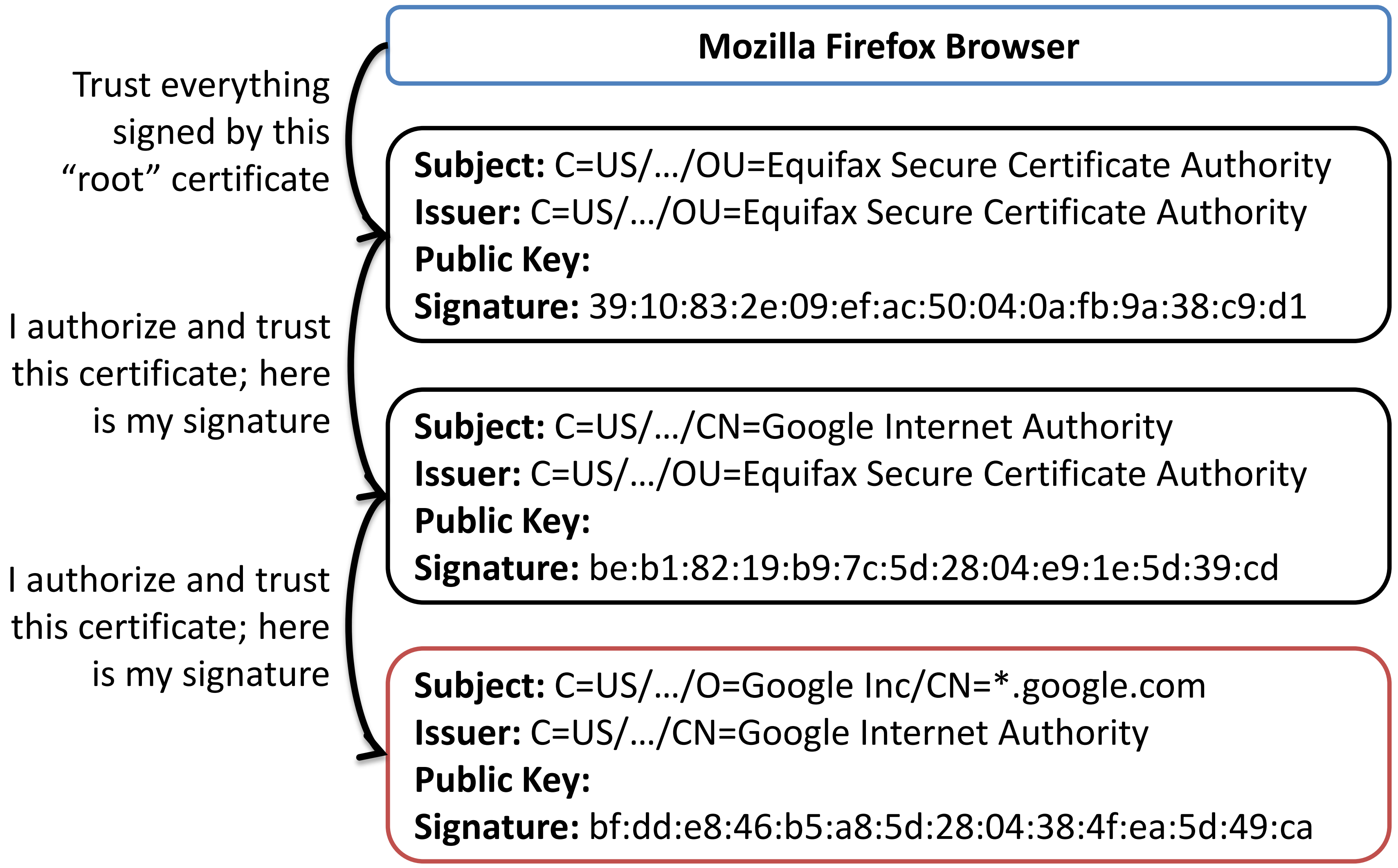
SSL/TLS Certificates

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

Certificate Chains



Mozilla Firefox Browser

Trust everything signed by this “root” certificate

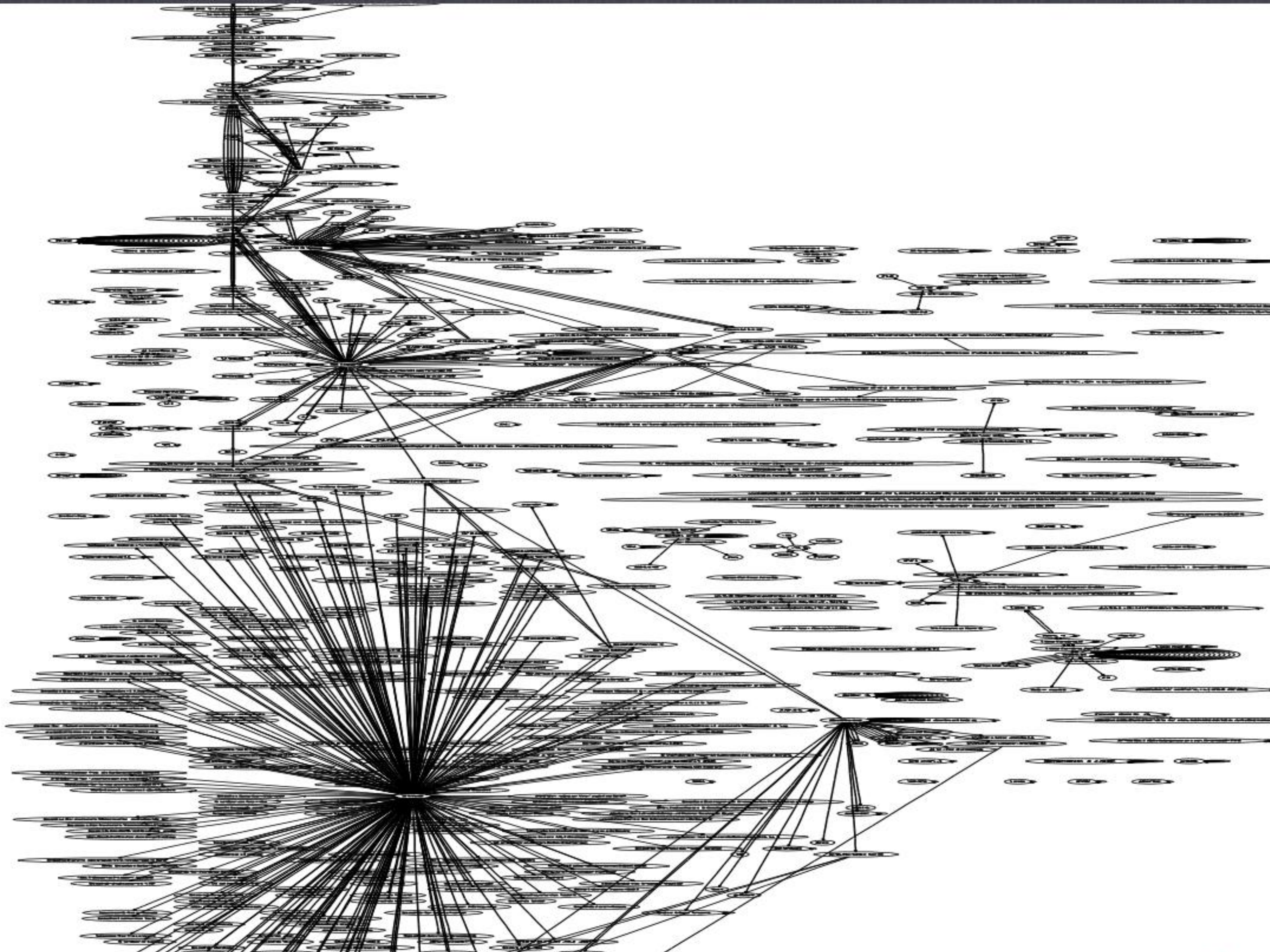
Subject: C=US/…/OU=Equifax Secure Certificate Authority
Issuer: C=US/…/OU=Equifax Secure Certificate Authority
Public Key:
Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:38:c9:d1

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*
- **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

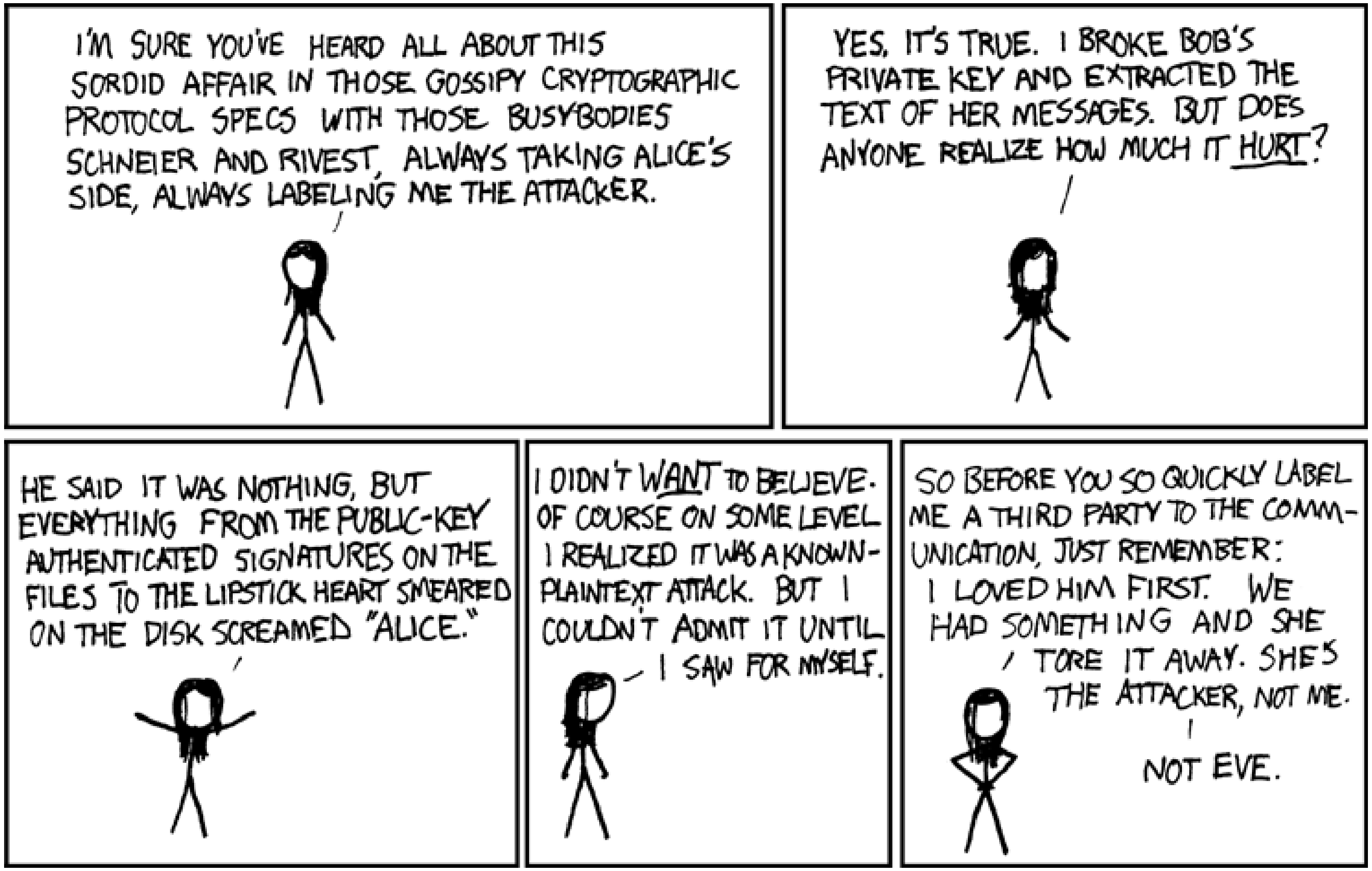
Don’t Roll Your Own!!

Questions?

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/

Thursday: 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.