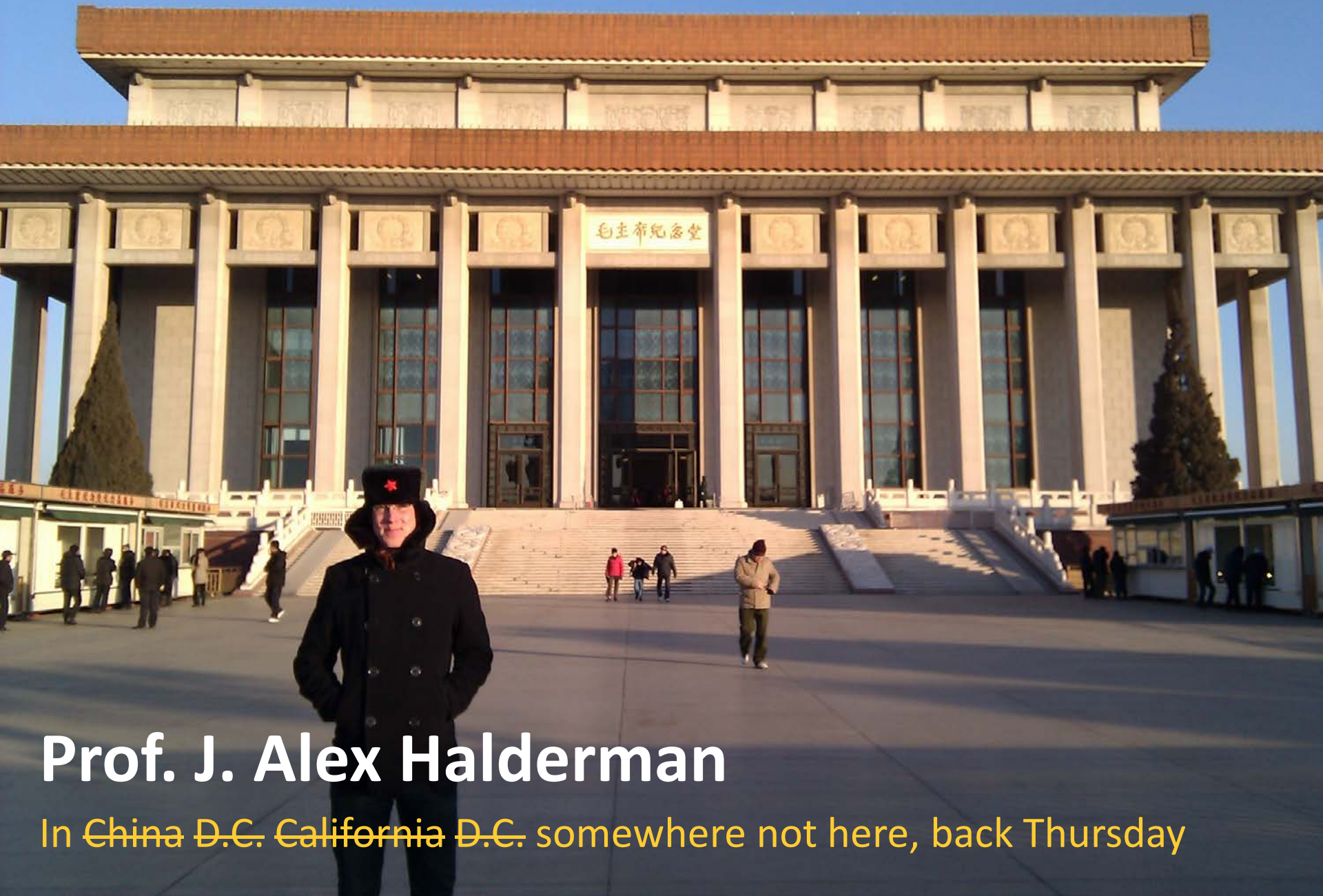


Essential Cryptography

EECS588 Computer and Network Security

January 9 and 16, 2014

The Itinerant Professor



Prof. J. Alex Halderman

In ~~China D.C.~~ ~~California D.C.~~ somewhere not here, back Thursday

Communication

Course Web Site

<https://www.eecs.umich.edu/courses/eecs588/>

Email Course Staff (Alex, Zakir, Eric)

eecs588@umich.edu

Course Structure

Readings (15%)

- Read classical and recent research papers in security. Come prepared to discuss in lecture.

Attack Presentation (30%)

- Implement a known attack from scratch and present (e.g. return oriented programming)

Research Project (50%)

- Investigate new attack, or defense, or analysis. Write workshop quality research paper.

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

Today's Class

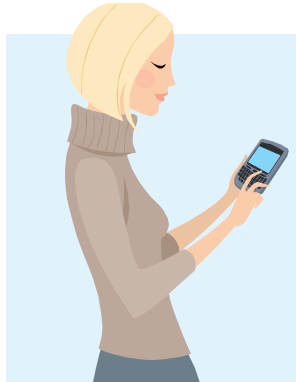
Essential Cryptography, Part 1

- Hash Functions
- Message-Authentication Codes
- Generating Random Numbers
- Block Ciphers

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

The Attacker



Alice

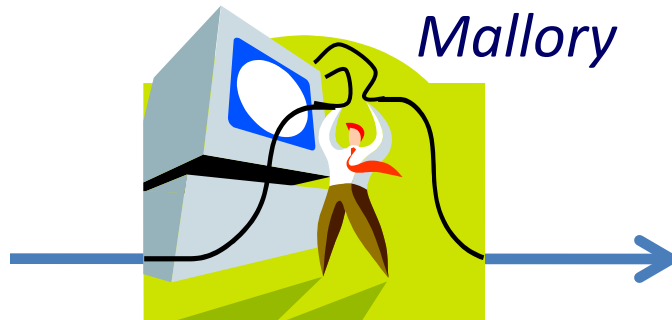
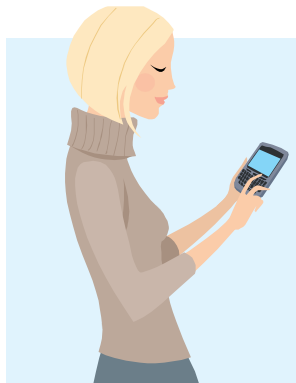


Eve

Passive Eavesdropper



Bob



Mallory

Man-in-the-Middle



Building a Secure Channel

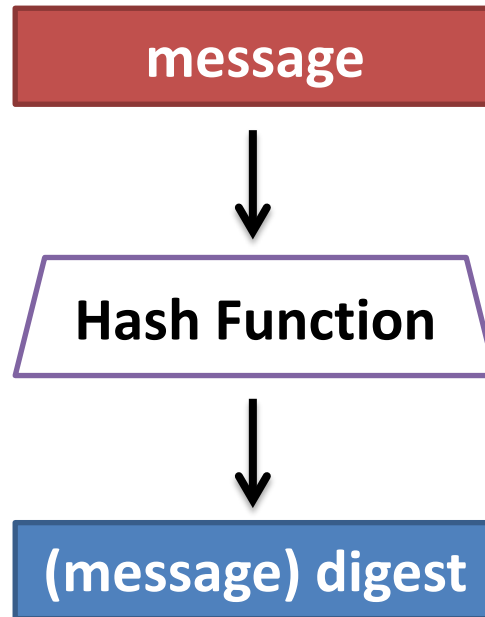
Confidentiality → Symmetric Ciphers

Integrity → Message Authentication Codes (MACs)

Authentication → Public Key Cryptography

Hash Functions

- Theoretically: random mapping from *any input* to a set of output



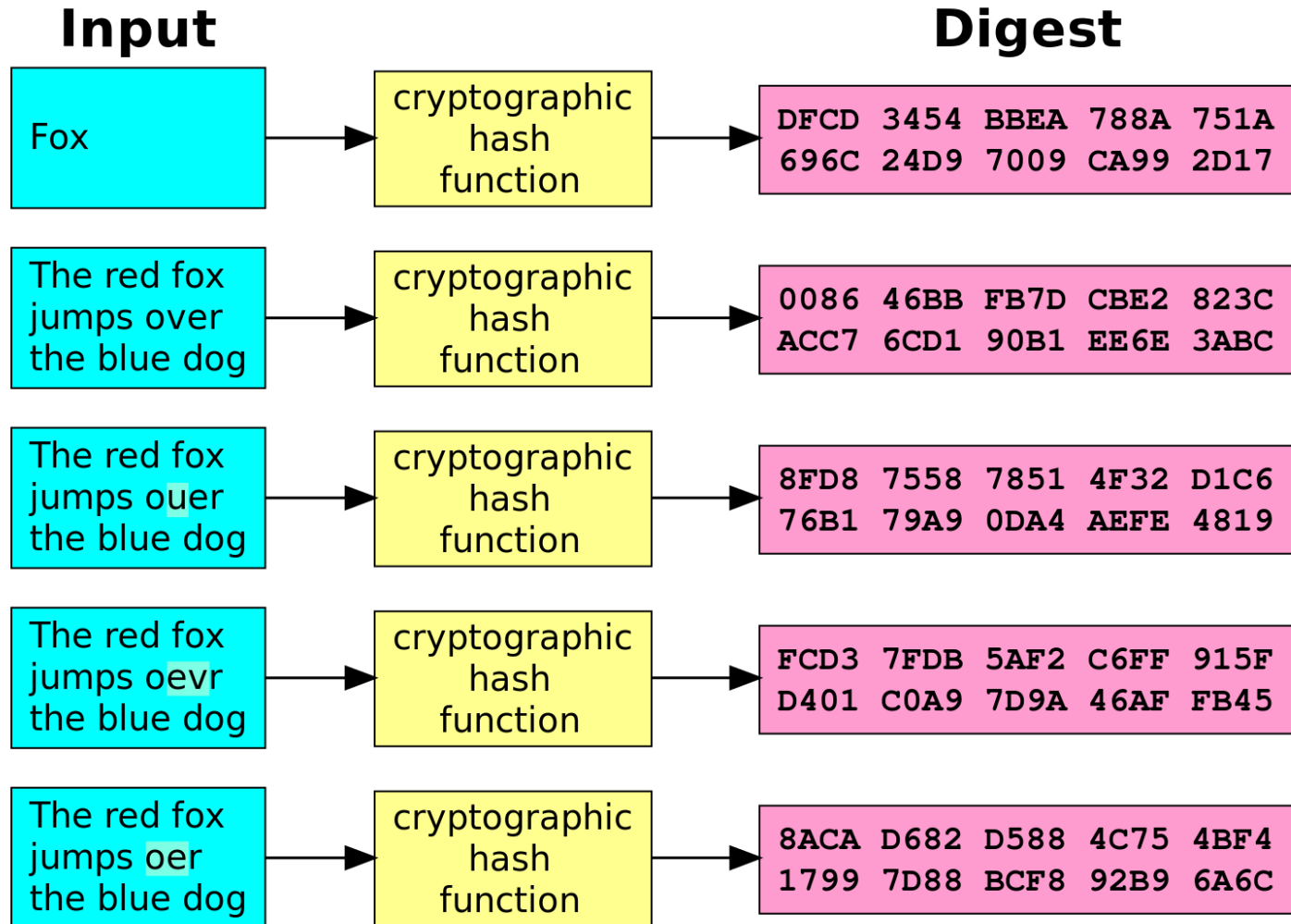
Ideal Cryptographic Hash Function

1. Easy to compute $H(m)$ for all m
2. Infeasible to compute m given $H(m)$
3. Infeasible to modify m without changing $H(m)$
4. Infeasible to find 2 messages with same hash

Hash Function Requirements

- Pre-Image Resistance
 - Given $h(x)$, infeasible to find x
- Second- Pre-image Resistance
 - Given m_1 , find m_2 s.t. $h(m_1) = h(m_2)$
- Collision Resistance
 - Given nothing, find *any* $m_1 \neq m_2$ s.t. $h(m_1) = h(m_2)$
 - Birthday Attack

Hash Functions

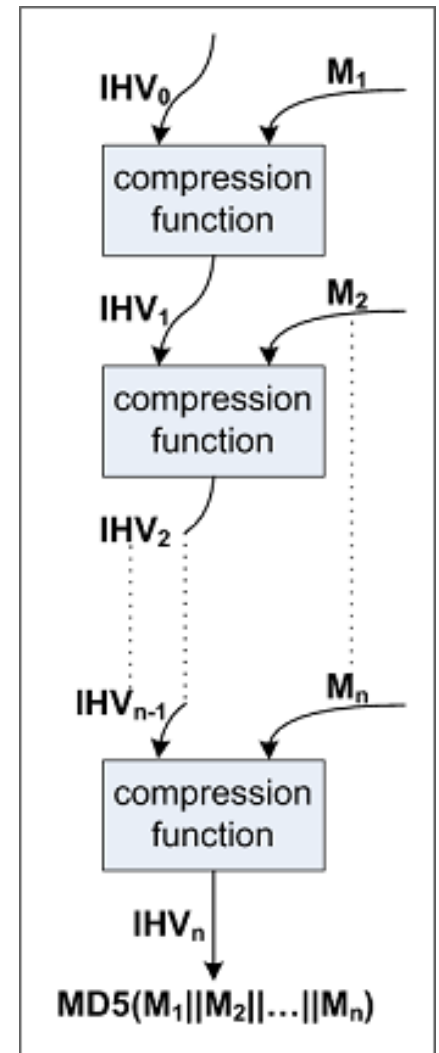


SHA Hash Functions

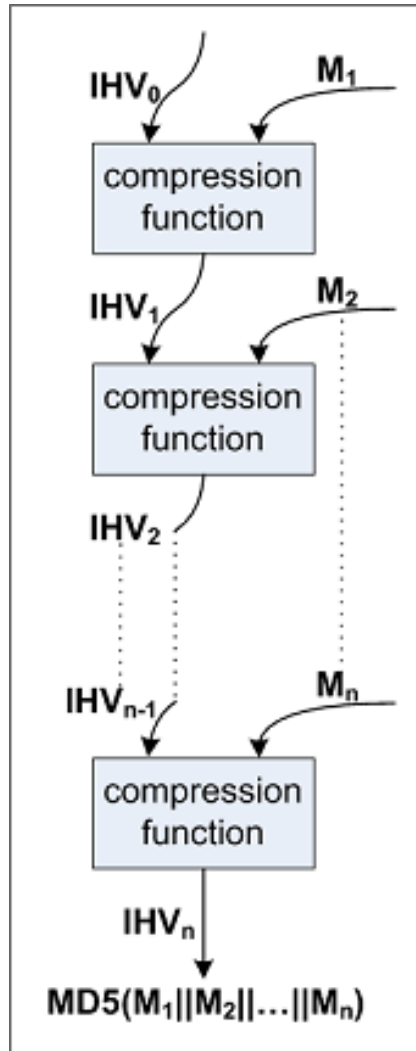
- SHA-1 – standardized by NIST in 1995
 - 160-bit output and internal state
 - 512-bit block size
- SHA-2 – extension published in 2001
 - 256 (or 512)-bit output and internal state
 - 512 (or 1024)-bit block size
- SHA-3 – chosen by NIST in 2012
 - 256 (512)-bit output
 - Different “sponge” construction

Block Chaining

- Most hash functions use block chaining to handle large unknown input sizes
- MD2, MD5, SHA-1, SHA-2
- Vulnerable to *Length Extension Attacks*



Length Extension Attacks



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

MD5 and SHA family all vulnerable!

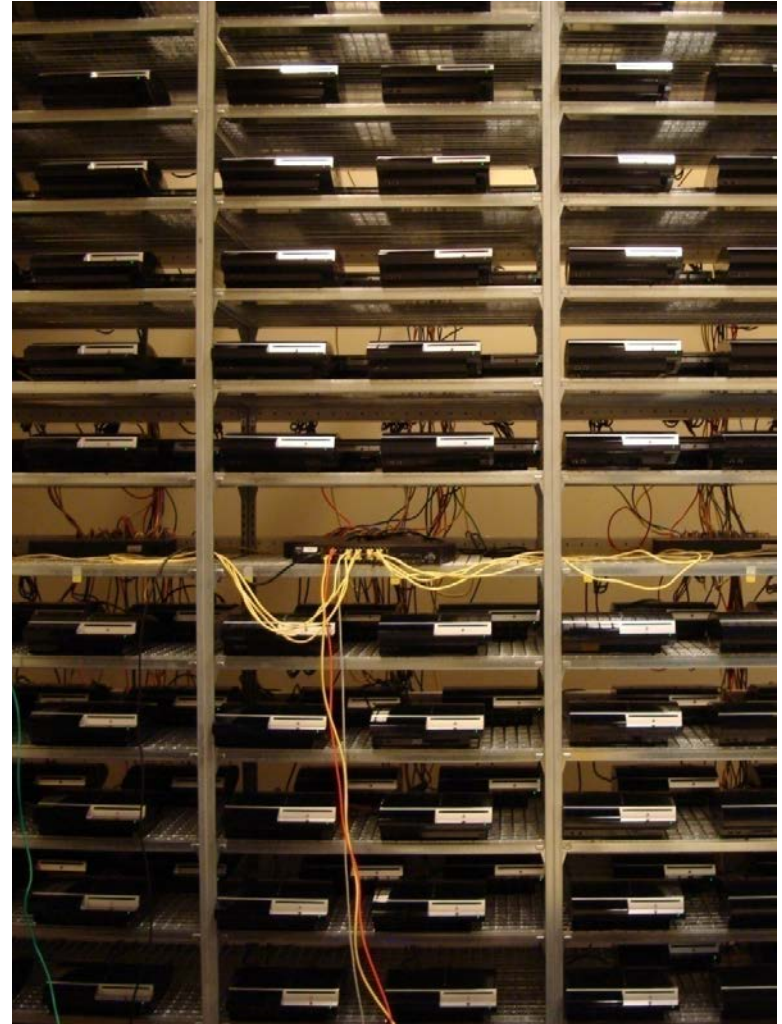
What does this let you do?

MD5 Hash Function

- Designed in 1992 by Ron Rivest
 - 128-bit output
 - 128-bit internal state
 - 128-bit block size
- Like most hash functions, uses block-chaining
- Weak! Do not use!

MD5 is Unsafe – Never use it!

- 1996: first flaws in 1996
- 2007: researchers demonstrated a collision
- Chaining allows chosen prefix attack
- 2008: used to fake SSL digital certificates



MD5 Collision

d131dd02c5e6eec4693d9a0698aff95c	2fcab5	8	712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325	7	1415a	085125e8f7cdc99fd91dbd
d8823e3156348f5bae6dacd436c919c6	dd53e2	b	487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080	a	80d1e	c69821bcb6a8839396f965
			2b6ff72a70

d131dd02c5e6eec4693d9a0698aff95c	2fcab5	0	712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325	f	1415a	085125e8f7cdc99fd91dbd
d8823e3156348f5bae6dacd436c919c6	dd53e2	3	487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080	2	80d1e	c69821bcb6a8839396f965
			ab6ff72a70

Both of these blocks hash to **79054025255fb1a26e4bc422aef54eb4**

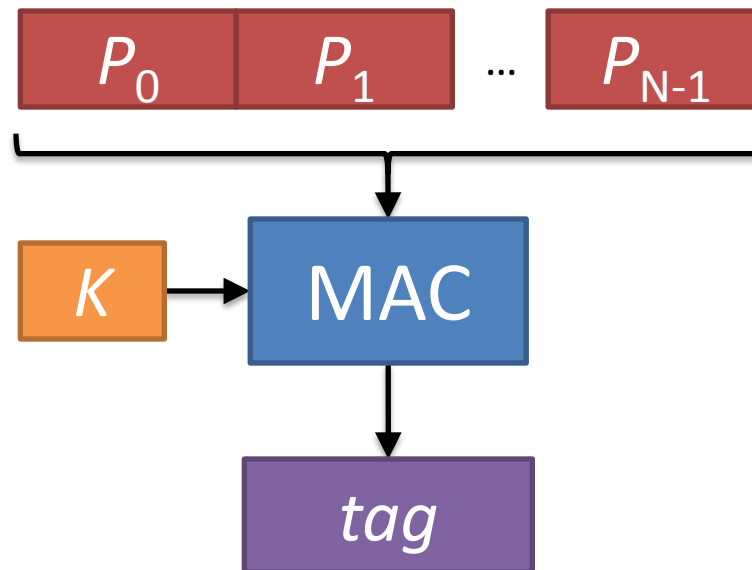
Is SHA-1 Safe?

- Significant cryptanalysis since 2005
- Improved attacks show complexity of finding a collision $< 2^{51}$ (ideally security would be 2^{80})
- Attacks only get better ...
- **Use SHA-256**

Message Authentication Codes

- **Prevents tampering with messages.**

Like a *family* of pseudorandom functions, with a key to select among them



HMAC Construction

Given a hash function H :

$$\text{HMAC}(K, m) = H((K \oplus \text{pad1}) \parallel H(K \oplus \text{pad2} \parallel m))$$

for constants pad1 and pad2

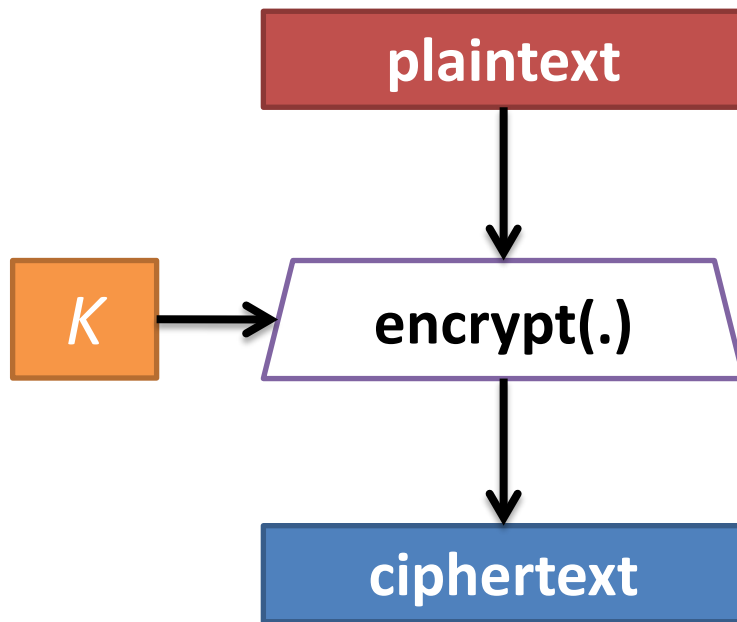
Provides nice, provable security properties

What should I use?

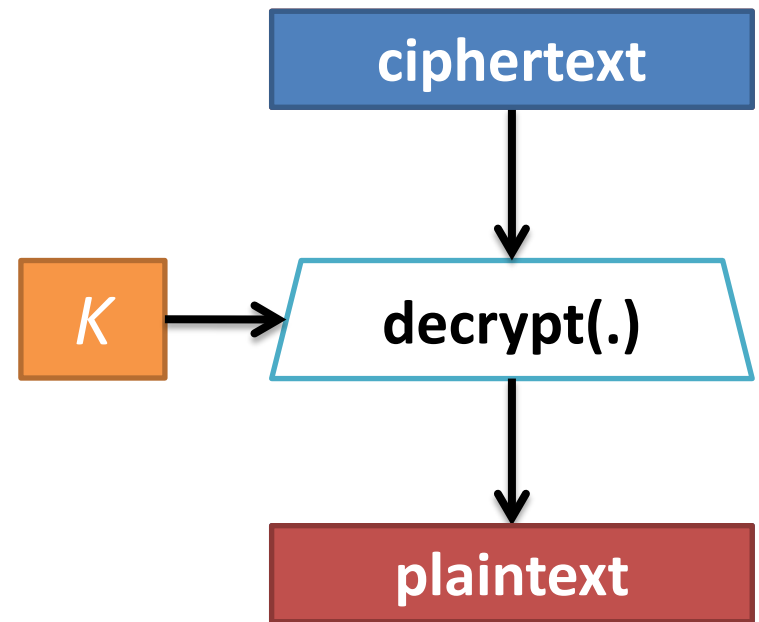
- **Hashes:** SHA-256 or SHA-3
- **HMACs:** HMAC-SHA256

Symmetric Key Encryption

Encryption



Decryption

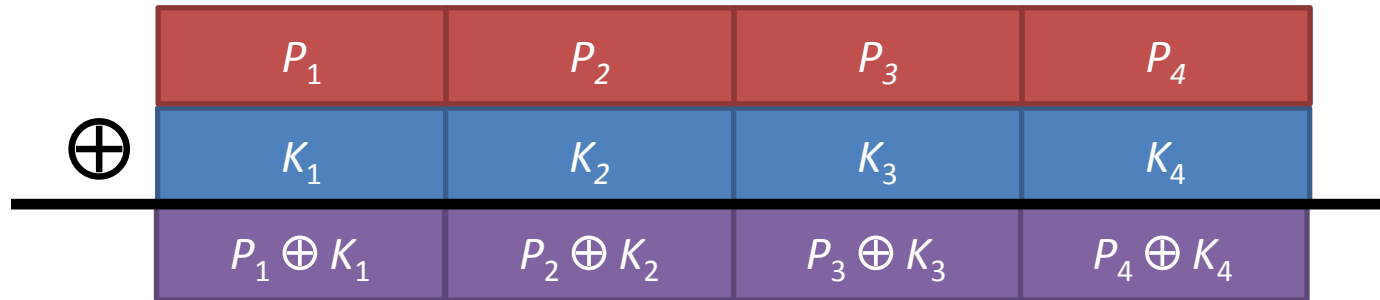


One-Time Pads

Provably secure encryption...

... that fails in practice.

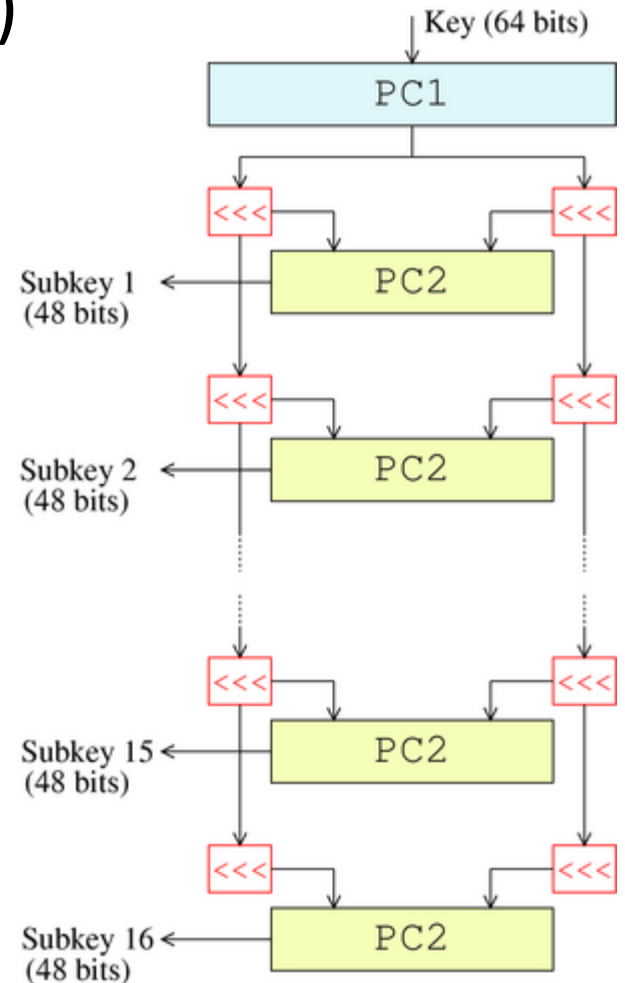
One-Time Pads



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

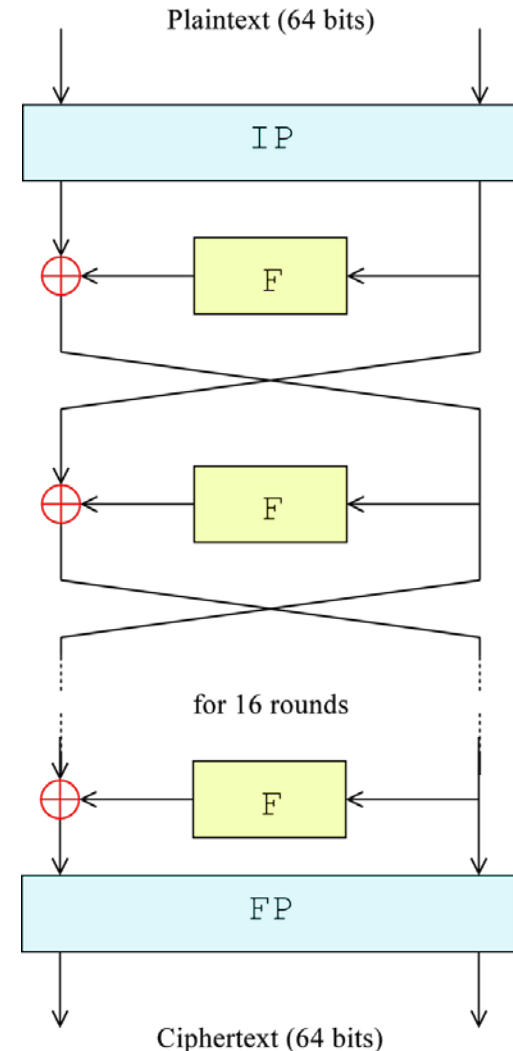
DES—Data Encryption Standard

- 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:



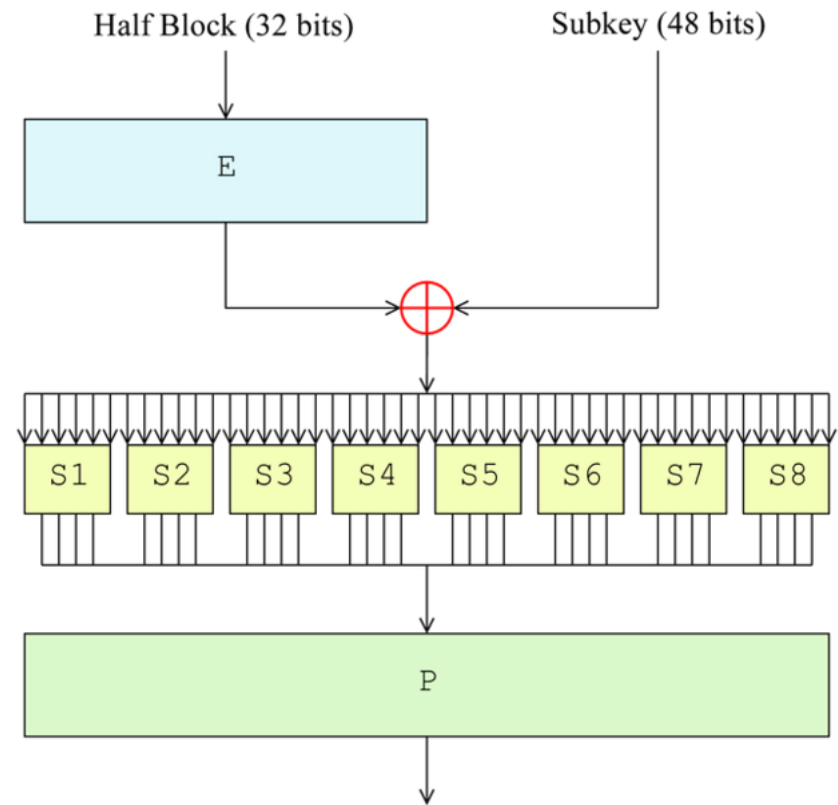
DES Encryption

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



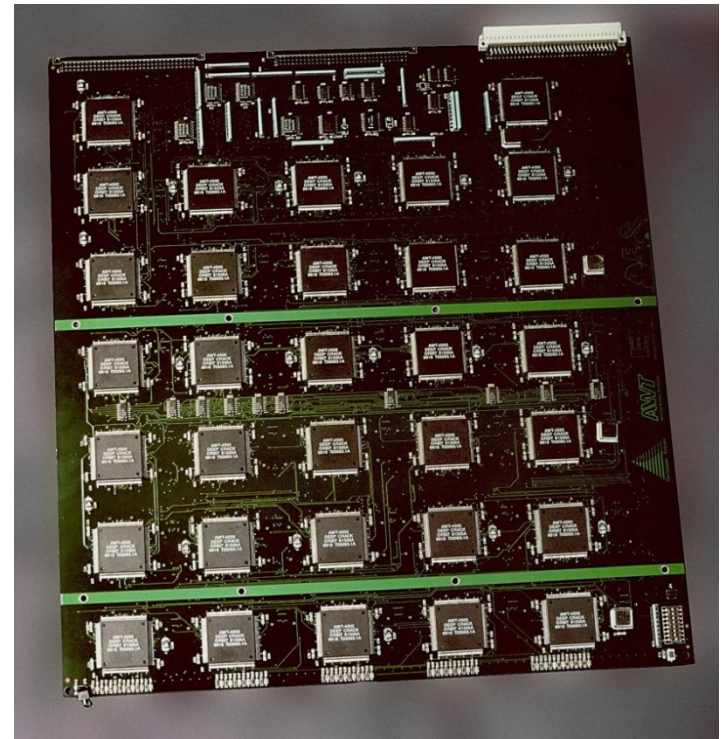
DES Feistel Function

- In each round:
 - Expansion Permutation E
32 \rightarrow 48 bits
 - S-boxes (“substitution”)
replace 6-bit values
 - Fixed Permutation P
rearrange the 32 bits



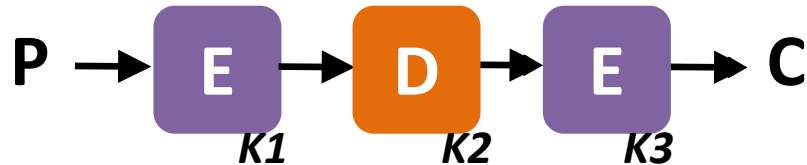
DES is Unsafe – Don't Use It!

- 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

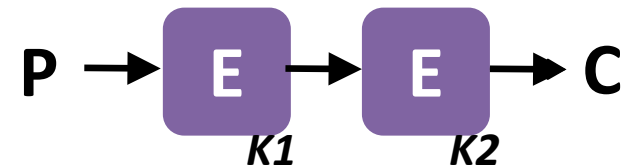
- $E_{K_1, K_2, K_3}(P) = E_{K_3}(D_{K_2}(E_{K_1}(P)))$



- Key options:
 - Option 1: independent keys ($56 \times 3 = 168$ bit key)
 - Option 2: $K_1 = K_3$ ($56 \times 2 = 112$ bit key)
 - Option 3: $K_1 = K_2 = K_3$ (Backward-compatible DES)
- What happened to 2DES?

2DES: Meet-in-the-middle attack

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



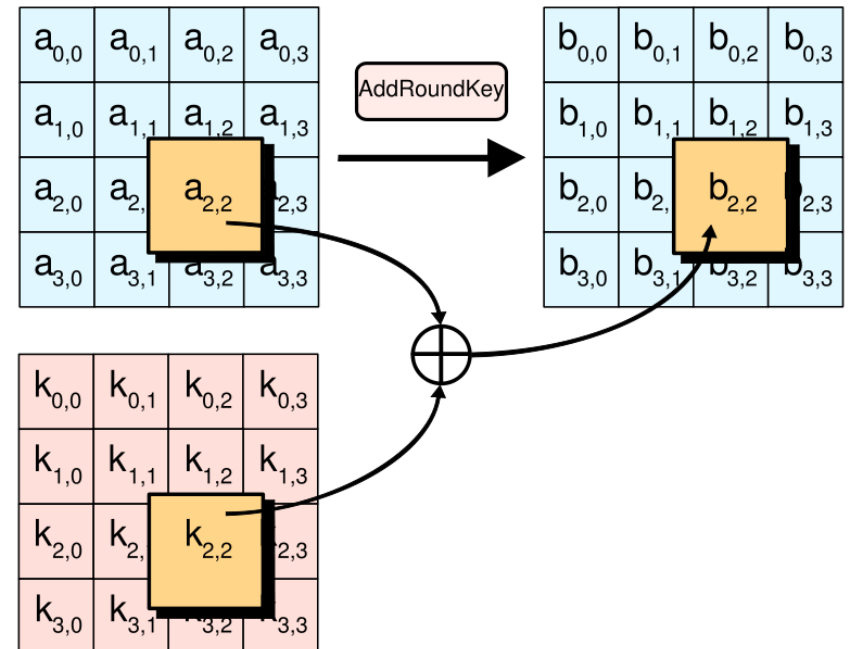
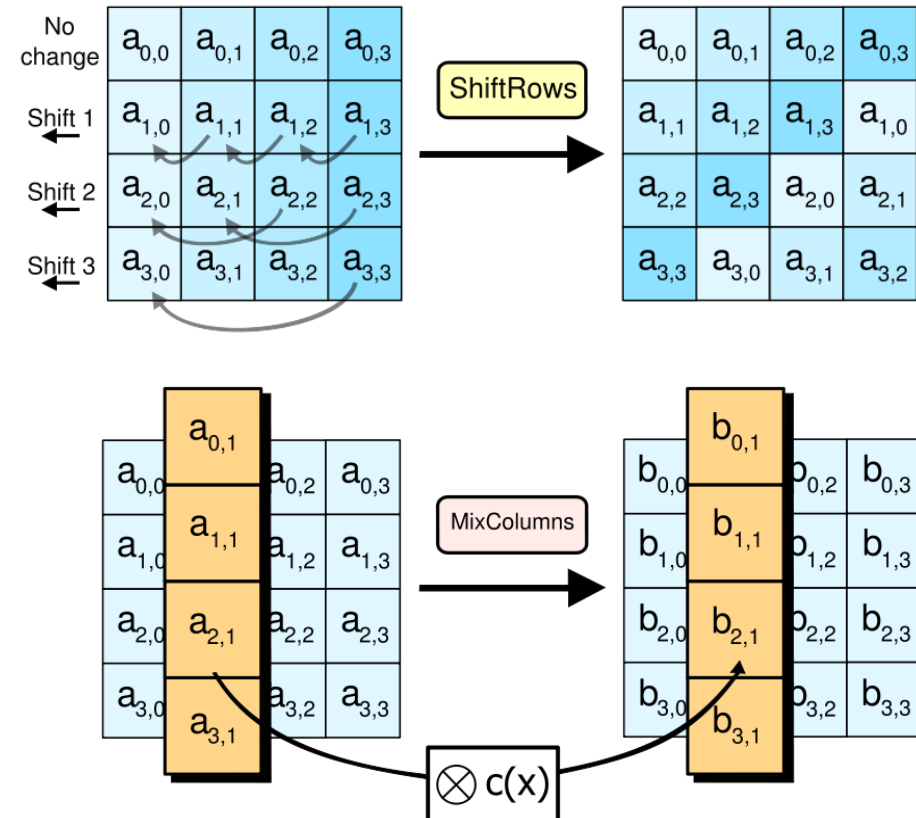
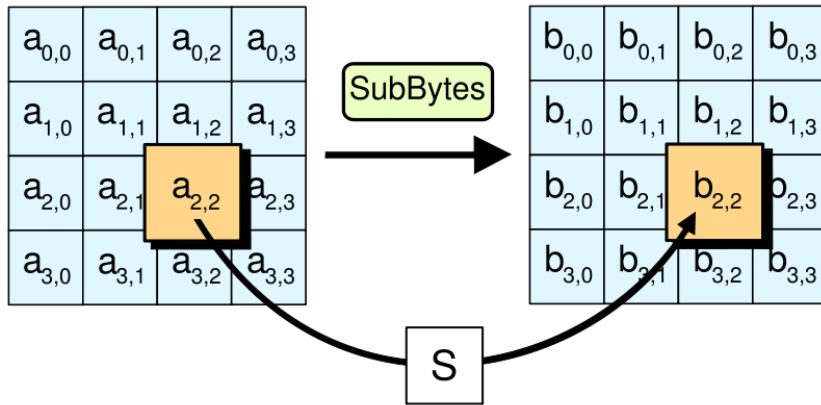
- Given P and $C = E_{K_2}(E_{K_1}(P))$, find both keys
 - For all K , generate $E_K(P)$ and $D_K(C)$
 - Find a match where $D_{K_2}(C) == E_{K_1}(P)$



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
- Most commonly used symmetric cipher

One round of AES-128

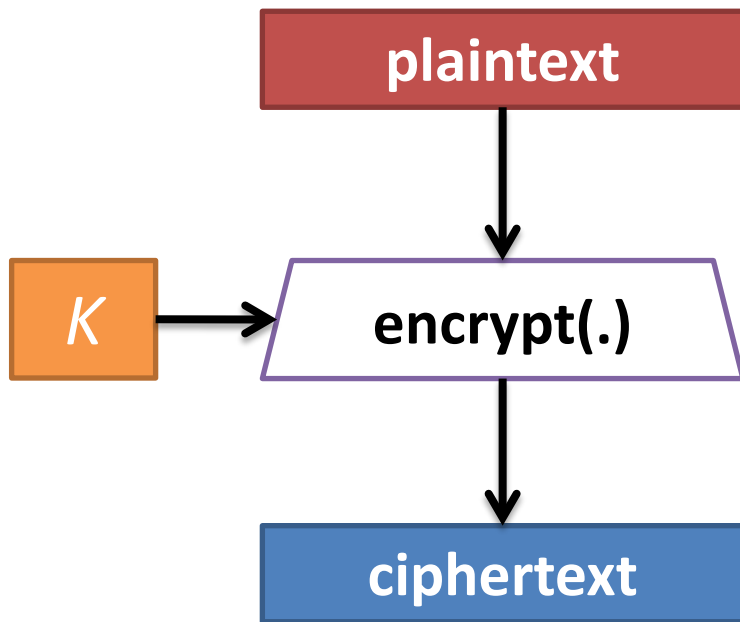


How Safe is AES?

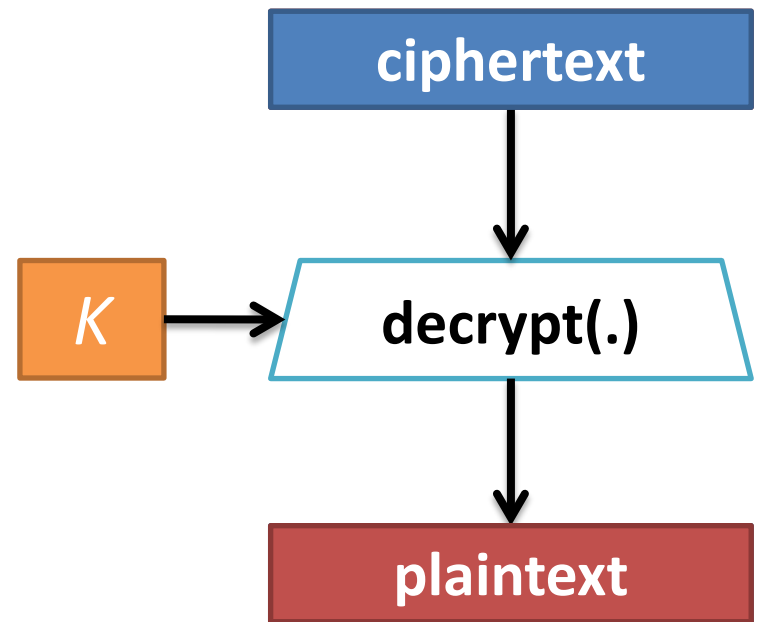
- Known attacks against 128-bit AES if reduced to 7 rounds (instead of 10)
- 128-bit AES very widely used, though NSA requires 192- or 256-bit keys for SECRET and TOP SECRET data
- What should you use?
 - Conservative answer: Use 256-bit AES

Block Ciphers (review)

Encryption

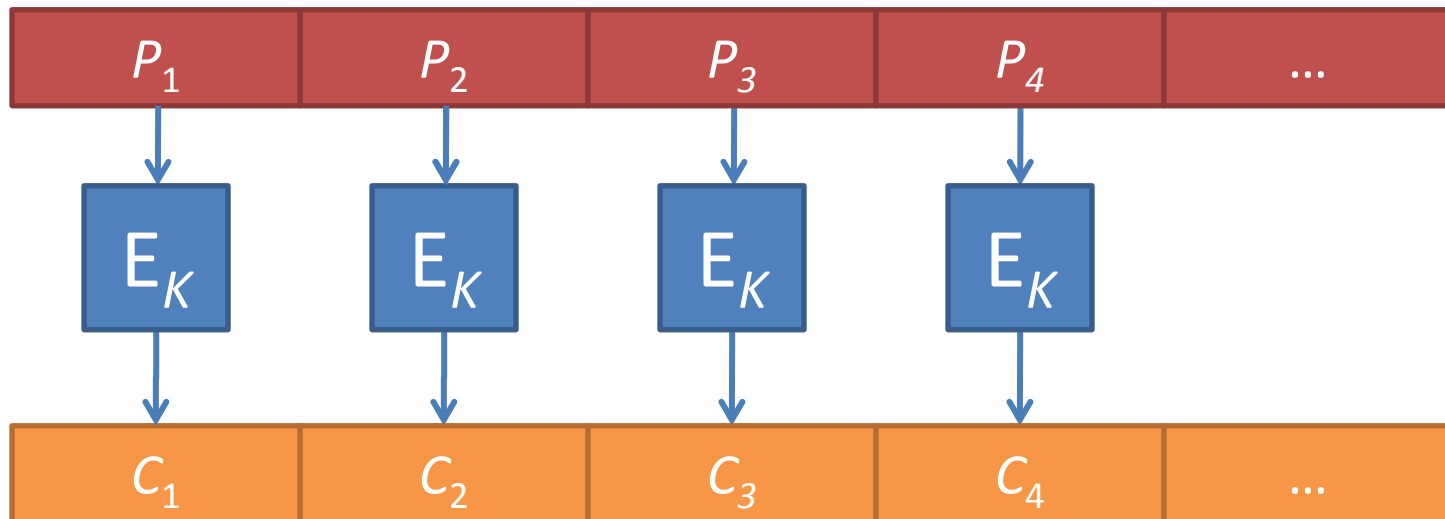


Decryption

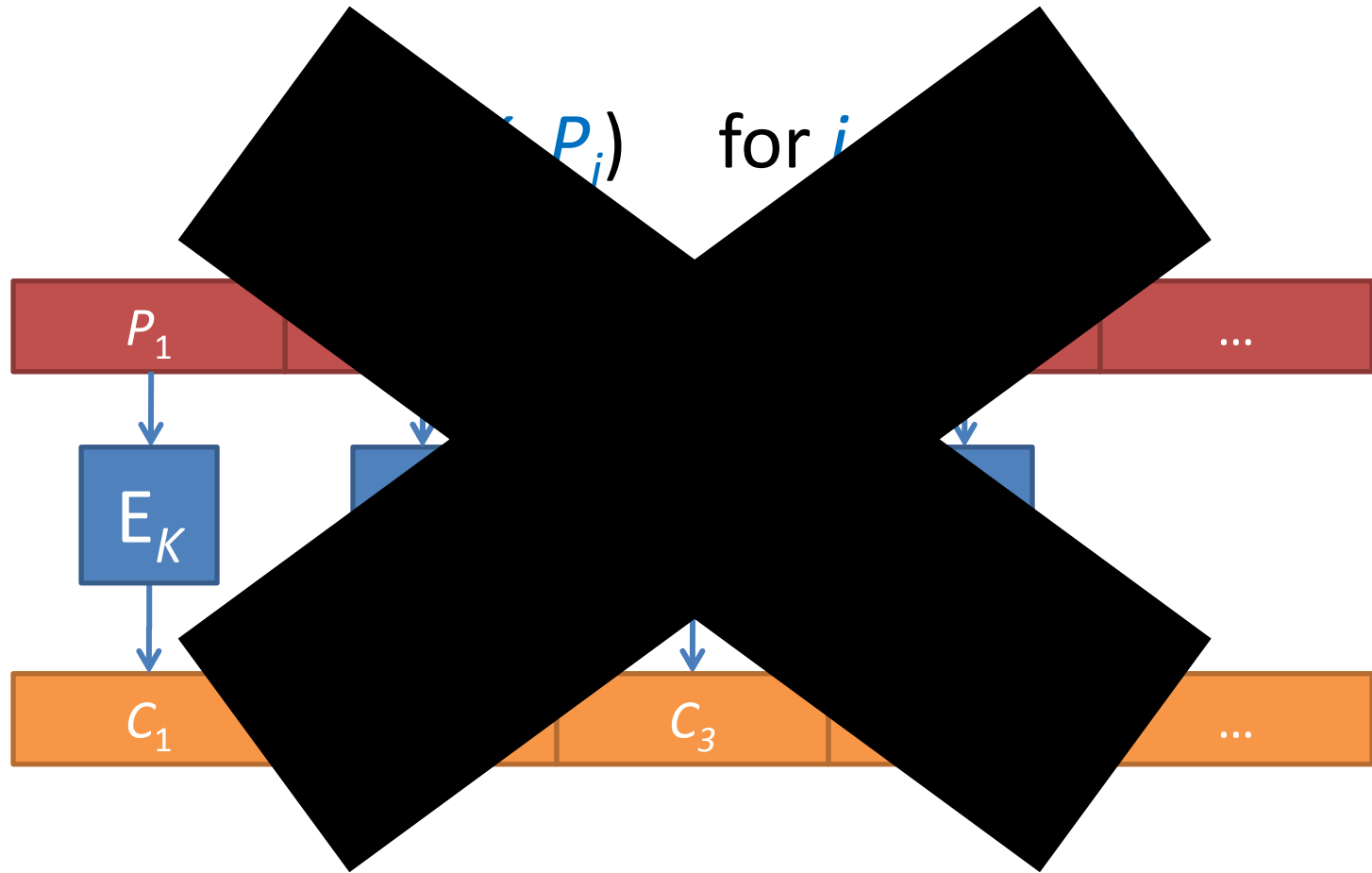


ECB – Electronic Codebook Mode

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

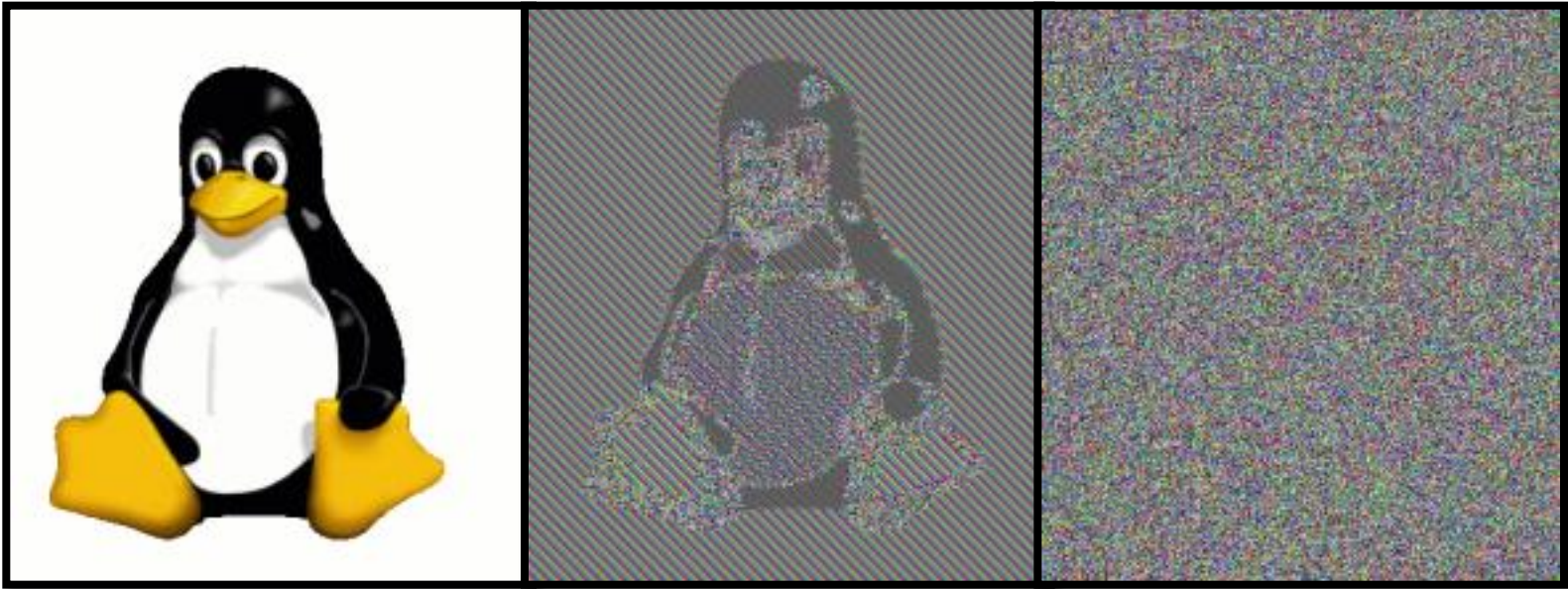


ECB – Electronic Codebook Mode



Why not ECB?

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



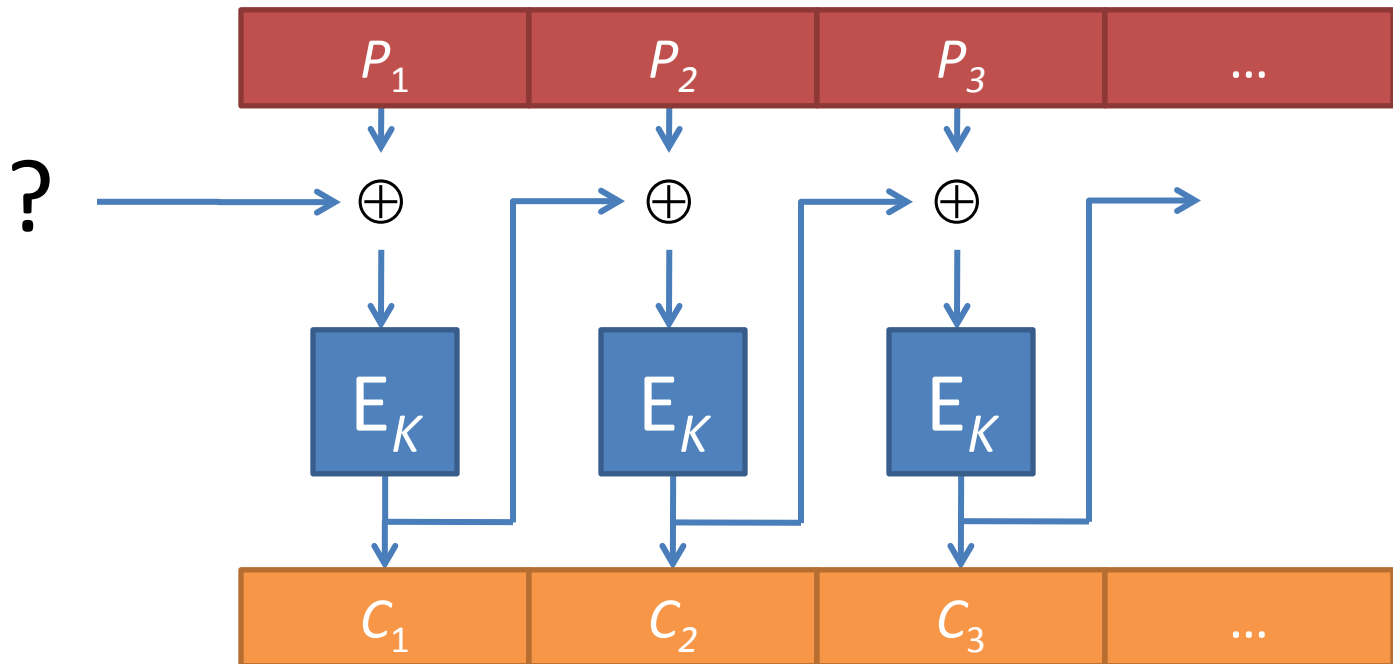
(plaintext)

(ECB mode)

(CBC mode)

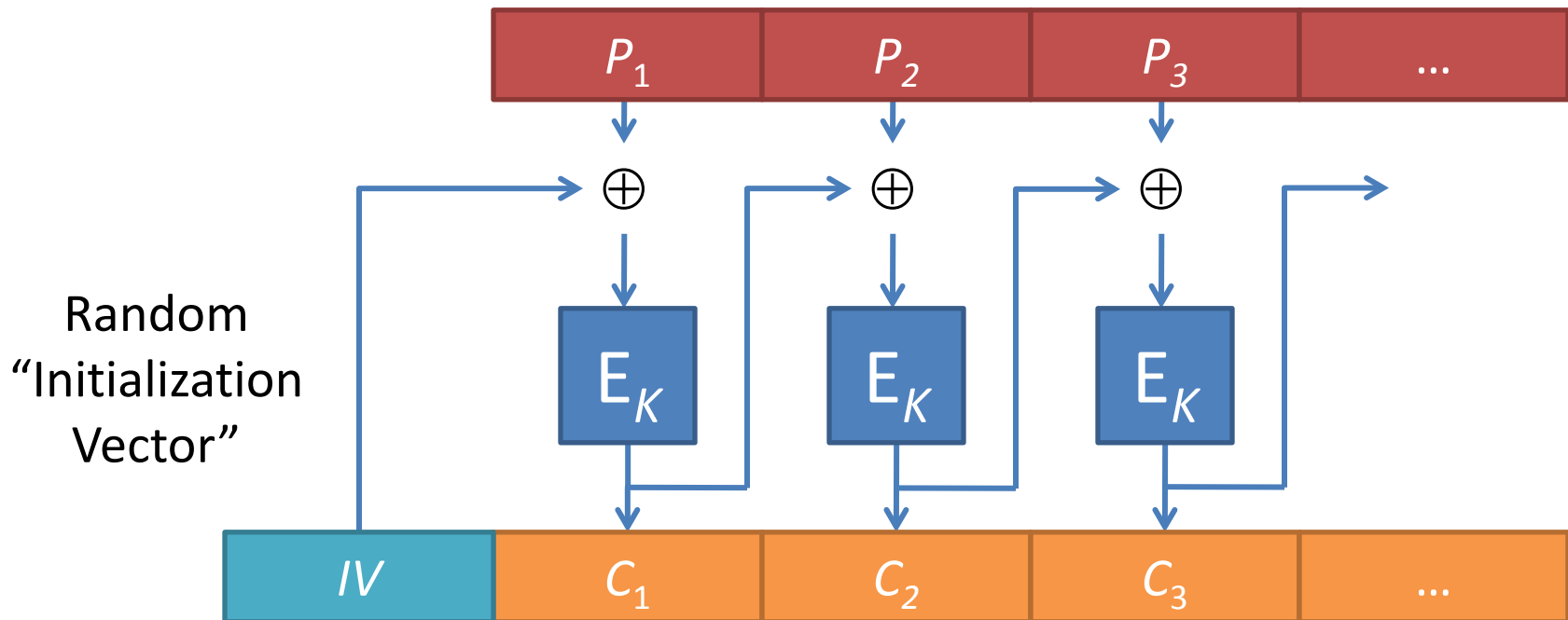
CBC: Cipher-Block Chaining Mode

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



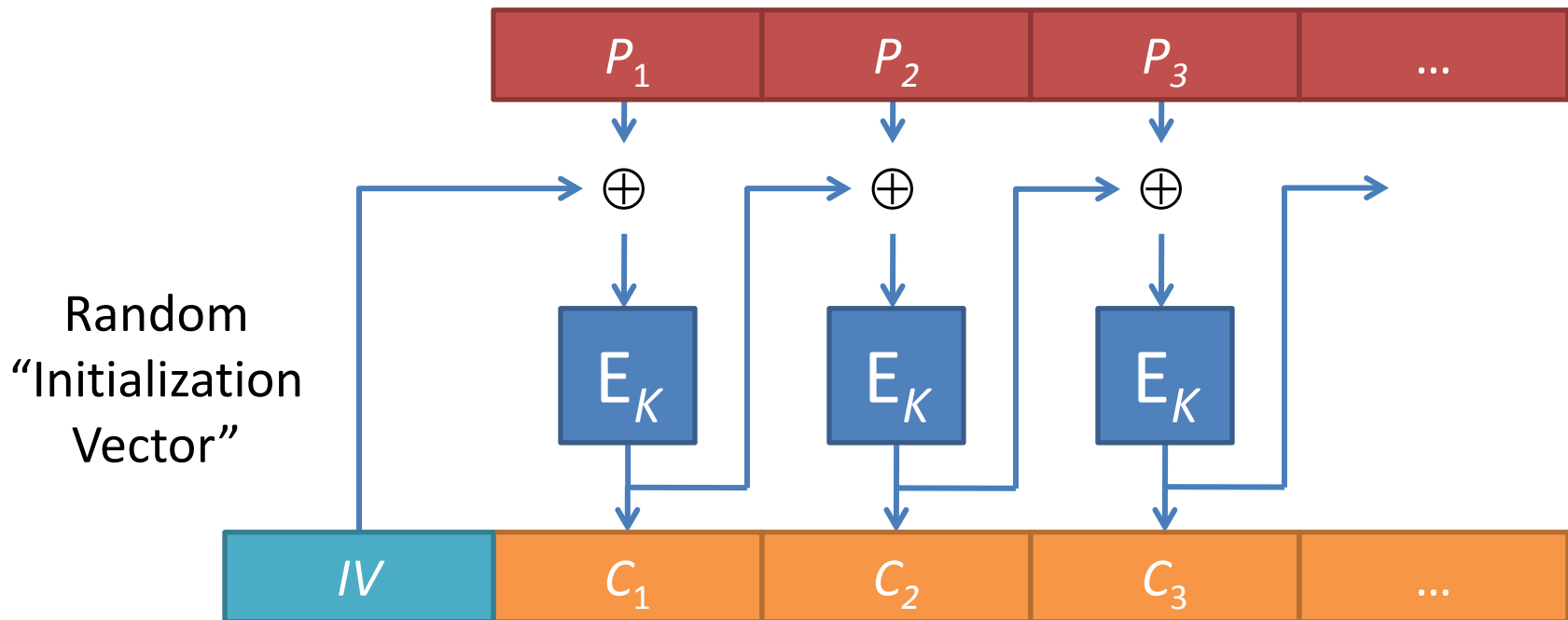
CBC: Cipher-Block Chaining Mode

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



CBC: Cipher-Block Chaining Mode

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



DO NOT REUSE INITIALIZATION VECTORS!!

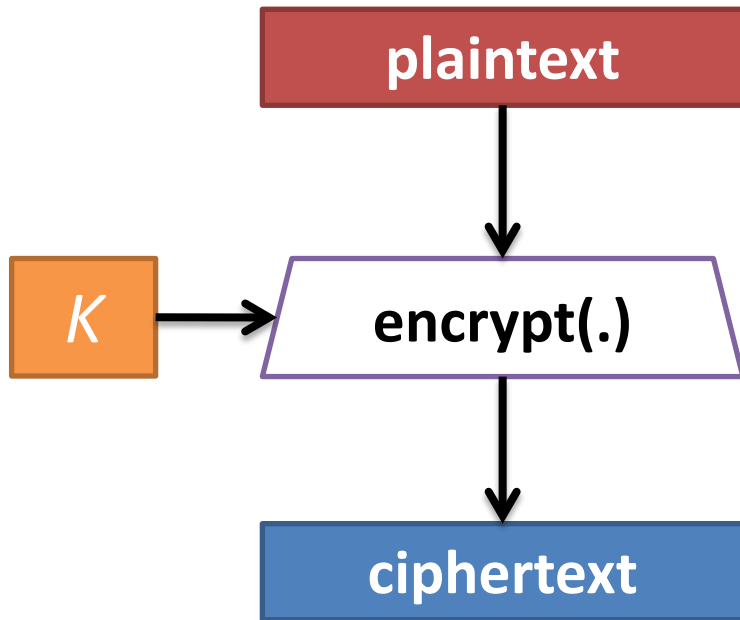
CTR: Counter Mode

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

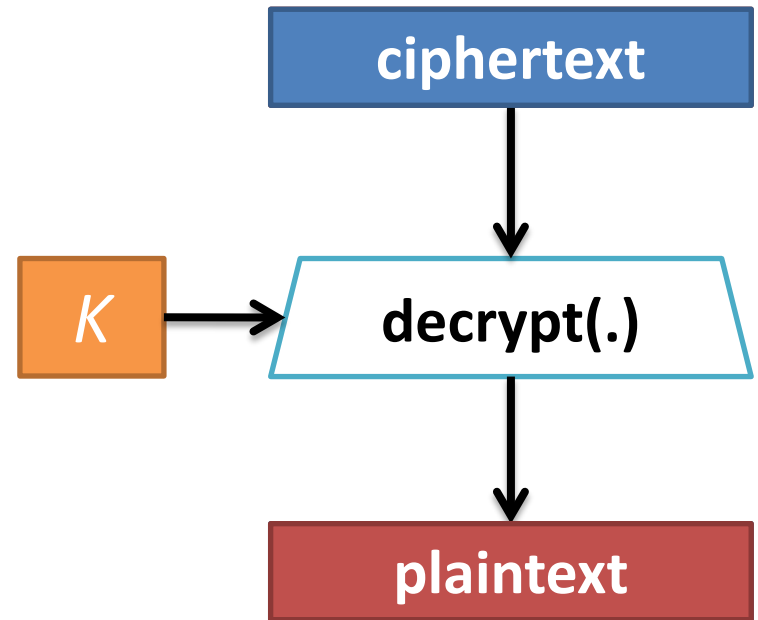
- Stream cipher construction
- Plaintext never passes through E
- Don't need to pad the message
- Allows parallelization and seeking
- Never reuse same $K + \textit{Nonce}$

Symmetric Key Encryption

Encryption



Decryption



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? (Asymmetric Cryptography)

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”)

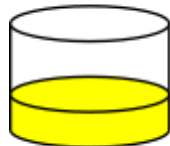
Diffie-Hellman Key Exchange

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 $g \in G$
2. We almost always just use \mathbb{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



+



=



Common paint

Secret colours

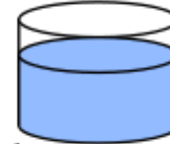
Bob



+

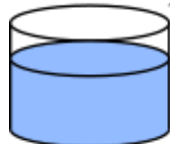


=



Public transport

(assume
that mixture separation
is expensive)



+



=



Common secret



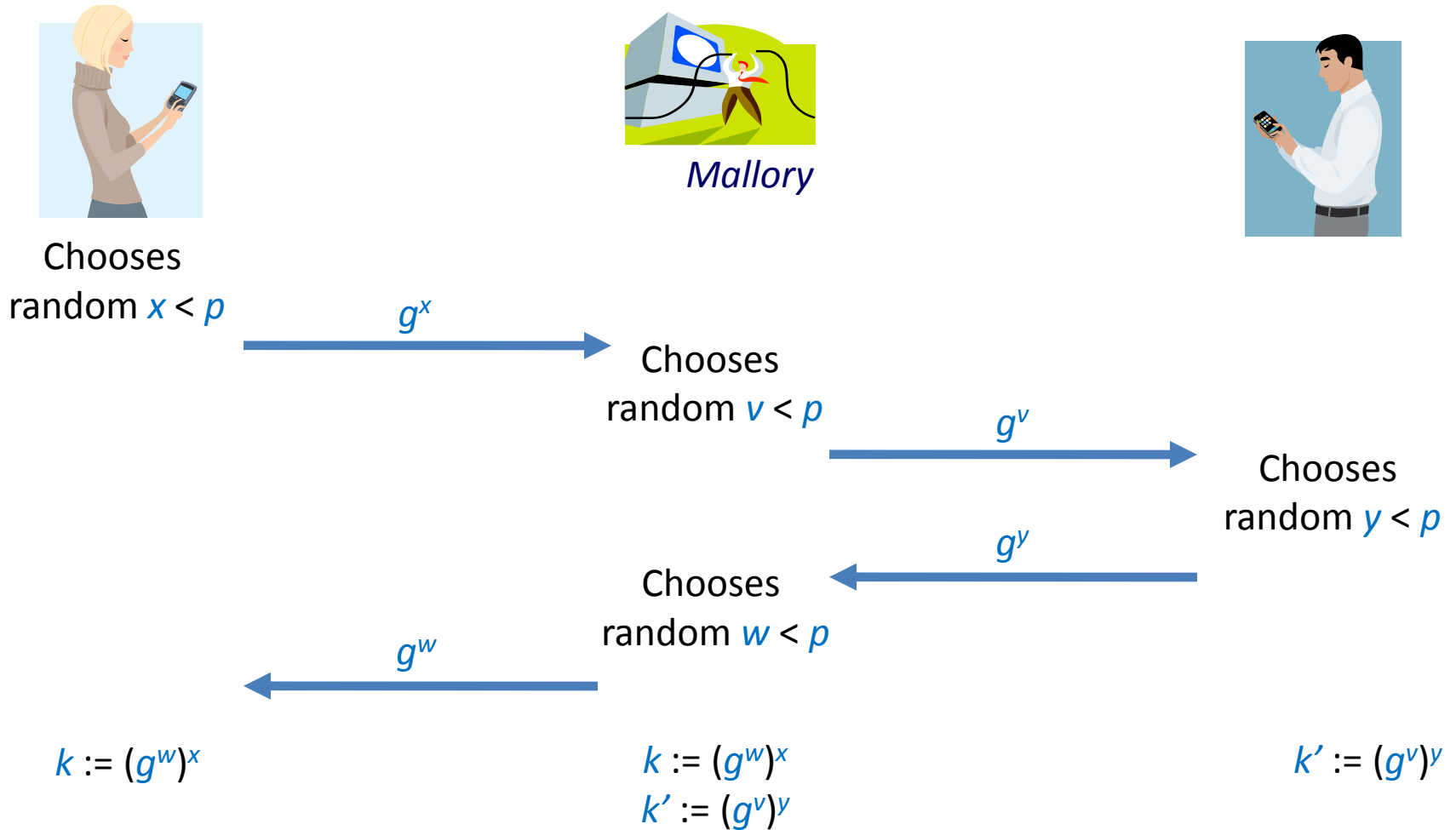
+



=



Attacking Diffie-Hellman (MITM)



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 $x^t = 1 \pmod{n}$

$e := [\text{small odd value}]$

public exponent

$d := e^{-1} \pmod{t}$

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

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

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

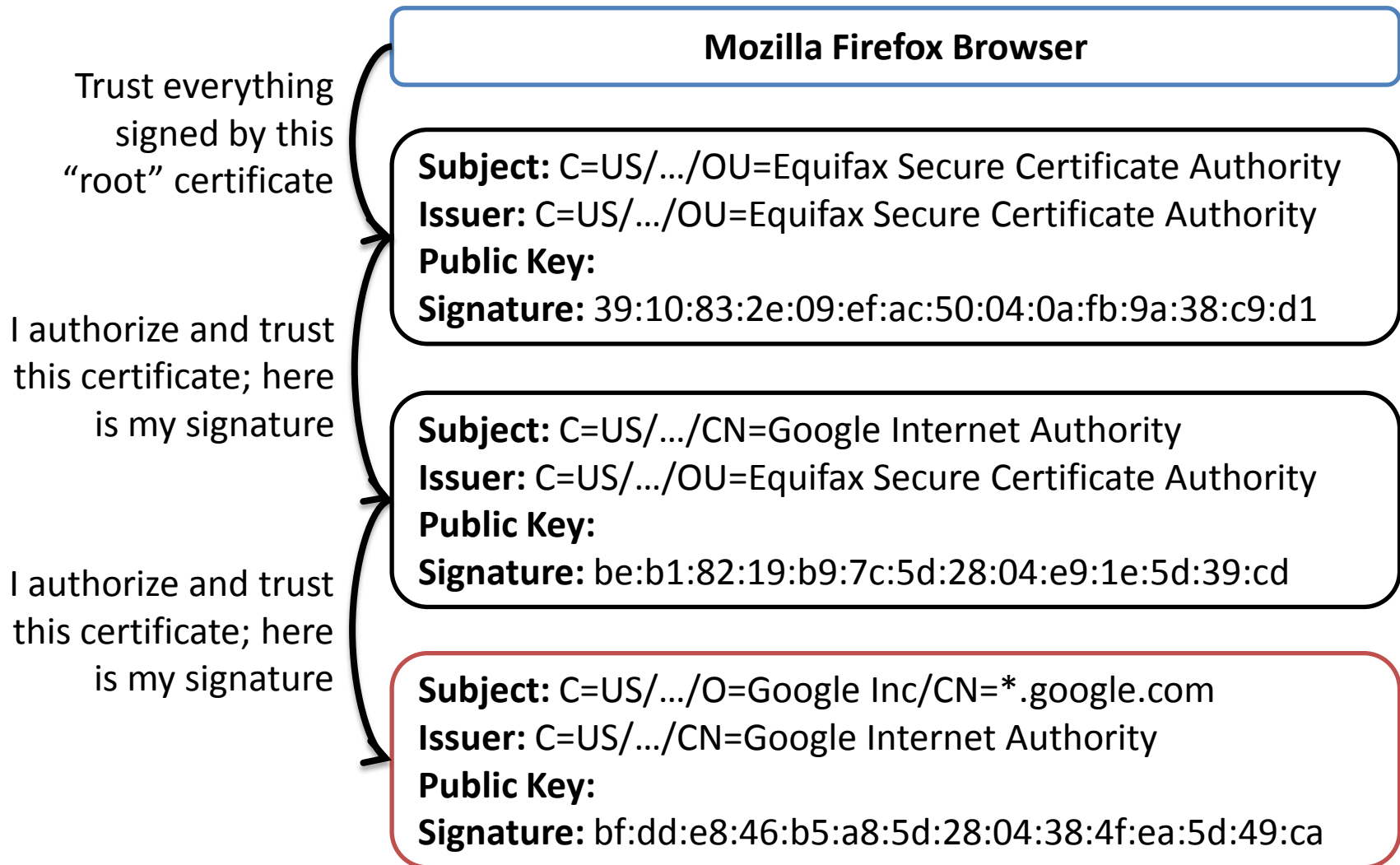
Signature Algorithm: sha1WithRSAEncryption

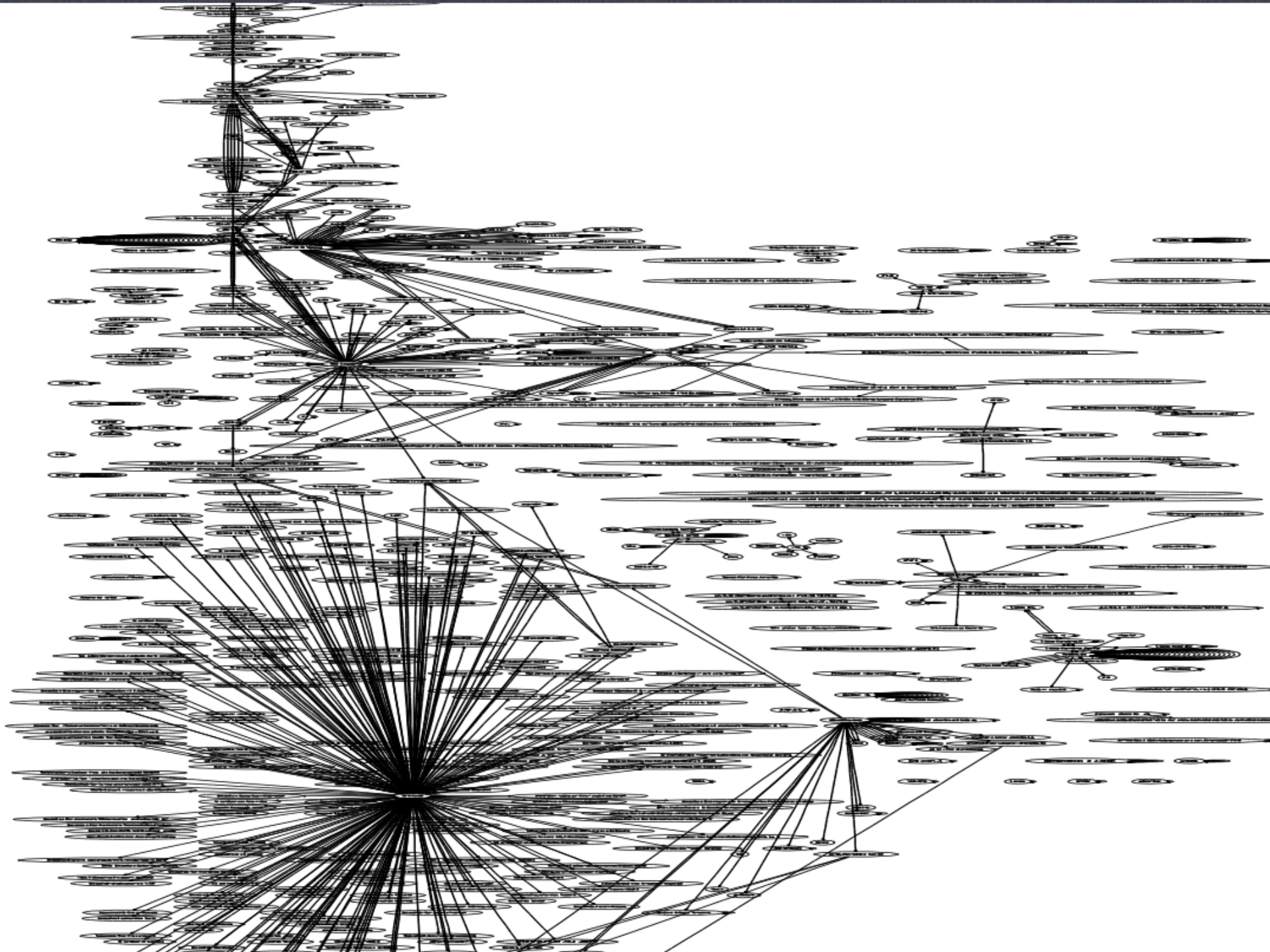
Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:1e5d:b5

Signatures on Certificates

- Utilize both public key cryptography and cryptographic hash functions
- Oftentimes see a signature algorithm such as **sha1WithRSAEncryption**
- **$\text{Encrypt}_{\text{PrivateKey}}(\text{SHA-1}(\text{certificate}))$**

Certificate Chains





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*

Case Study: SSL/TLS

- Arguably the most important (and widely used) cryptographic protocol on the Internet
- Almost all encrypted protocols (minus SSH) use SSL/TLS for transport encryption
- HTTPS, POP3, IMAP, SMTP, FTP, NNTP, XMPP (Jabber), OpenVPN, SIP (VoIP), ...

SSL vs. TLS

- SSL := Secure Sockets Layer (Netscape)
- TLS := Transport Layer Security (IETF)
- Terms are used interchangeably
- SSL 3.0 is predecessor to TLS 1.0

Browser TLS Support

Browser ↕	Platforms ↕	TLS 1.0 ↕	TLS 1.1 ↕	TLS 1.2 ↕
Chrome 0–22	Linux, Mac OS X, Windows (XP, Vista, 7) ^[a]	Yes	No	No
Chrome 22–	Linux, Mac OS X, Windows (XP, Vista, 7) ^[a]	Yes	Yes	No
Firefox 2–	Linux, Mac OS X, Windows (XP, Vista, 7)	Yes ^[34]	No ^[35]	No ^[36]
IE 1–7	Mac OS X, Windows (XP, Vista, 7) ^[b]	Yes	No	No
IE 8–	Windows 7 ^[b]	Yes	Yes	Yes
Opera 10–	Linux, Mac OS X, Microsoft Windows ^[c]	Yes	Yes, disabled	Yes, disabled
Safari 5–	Mac OS X, Windows (XP, Vista, 7) ^[d]	Yes	?	?

Where does TLS live?

Application (HTTP)

Transport (TCP)

Network (IP)

Data-Link (1gigE)

Physical (copper)



Client

Server

“the handshake”

Client

Server


Client Hello: Here's what I support and a *random*



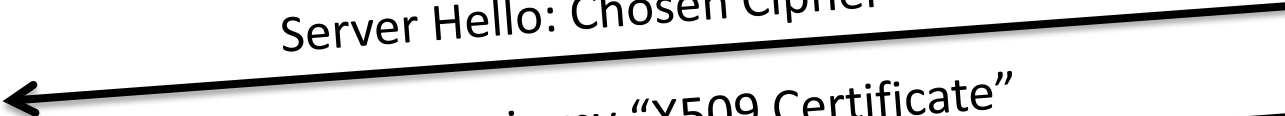
Client

Server

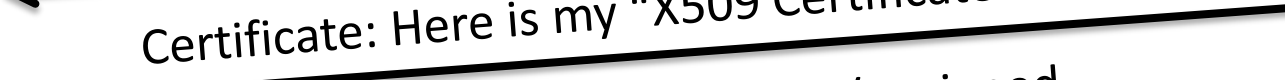
Client Hello: Here's what I support and a *random*



Server Hello: Chosen Cipher



Certificate: Here is my "X509 Certificate"

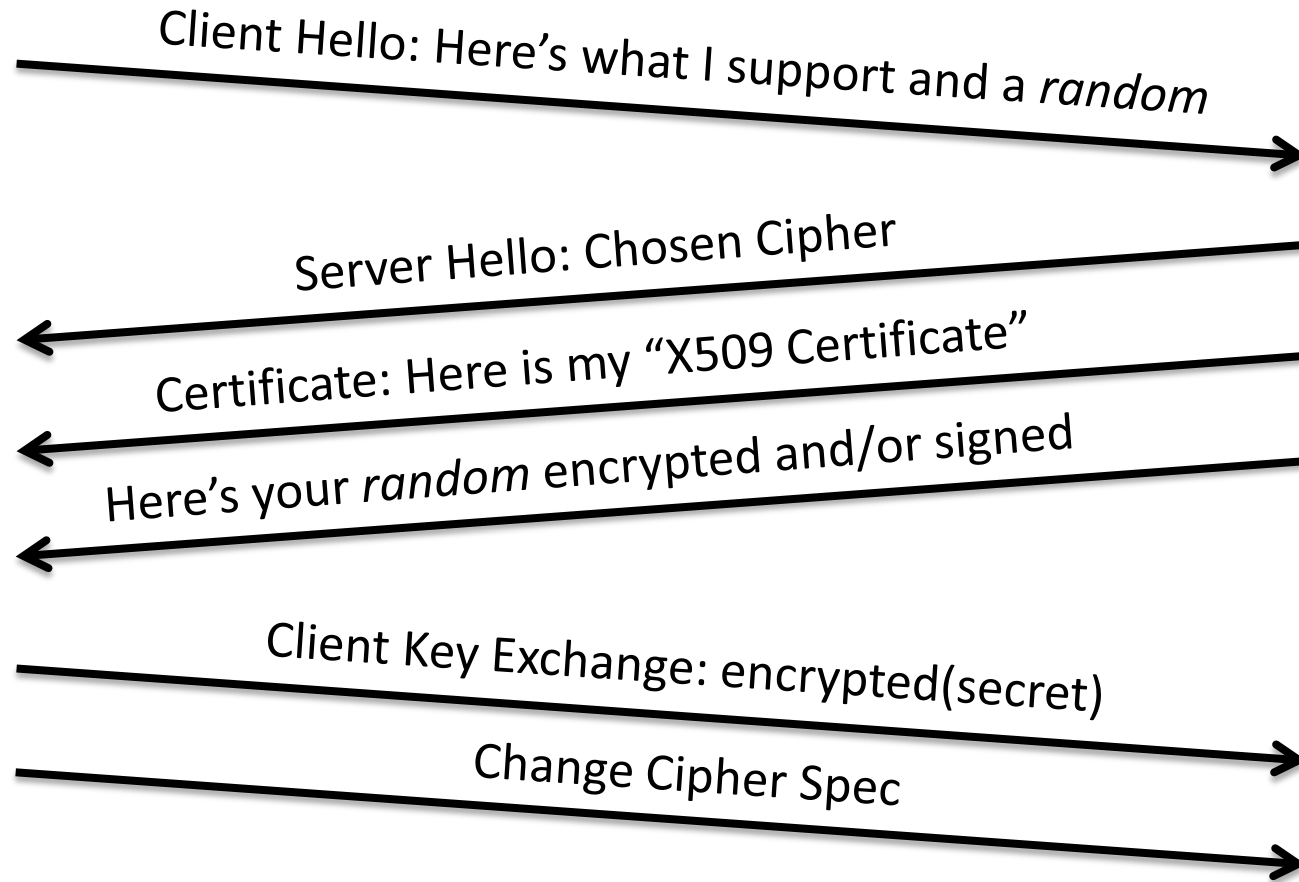


Here's your *random* encrypted and/or signed



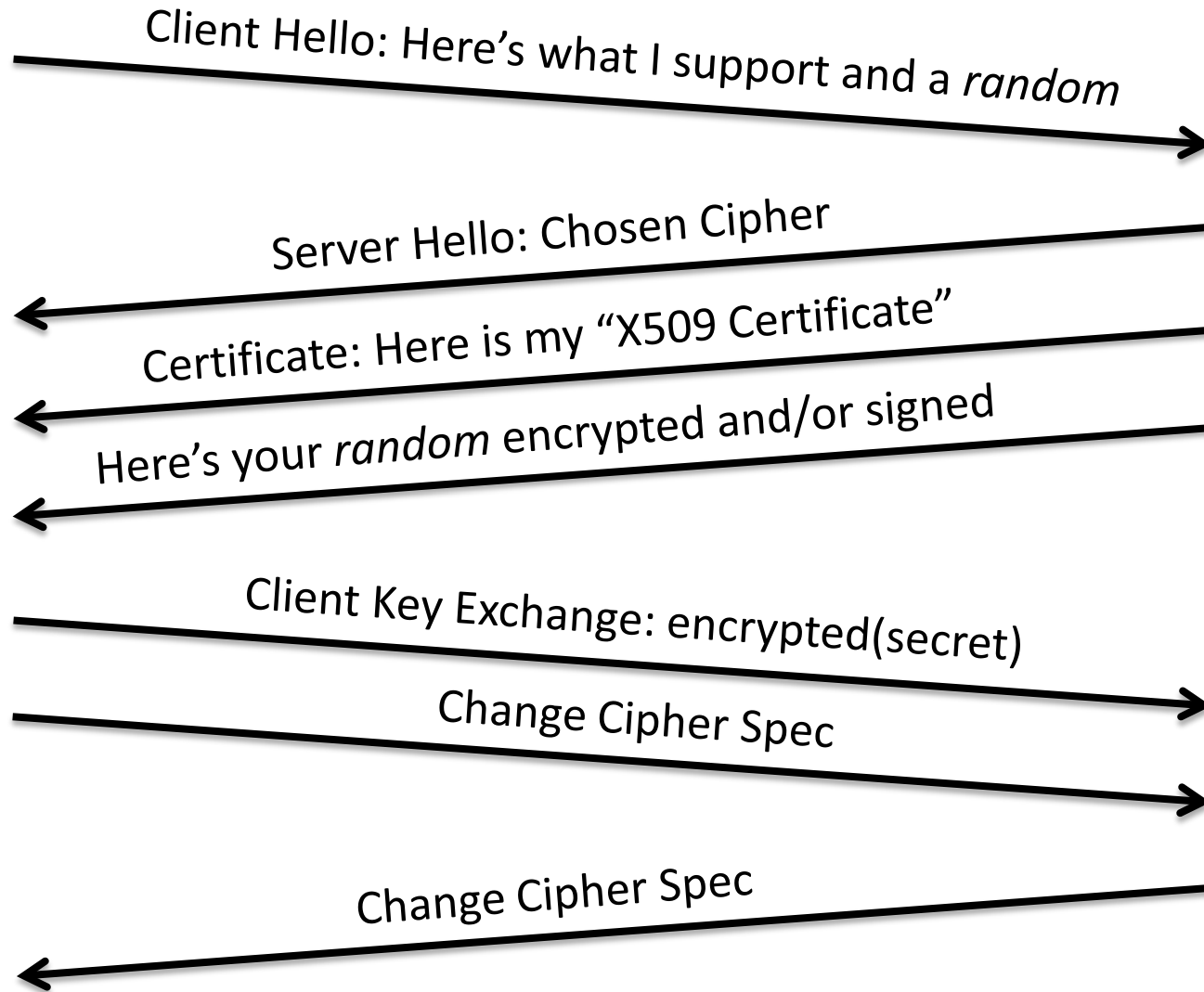
Client

Server



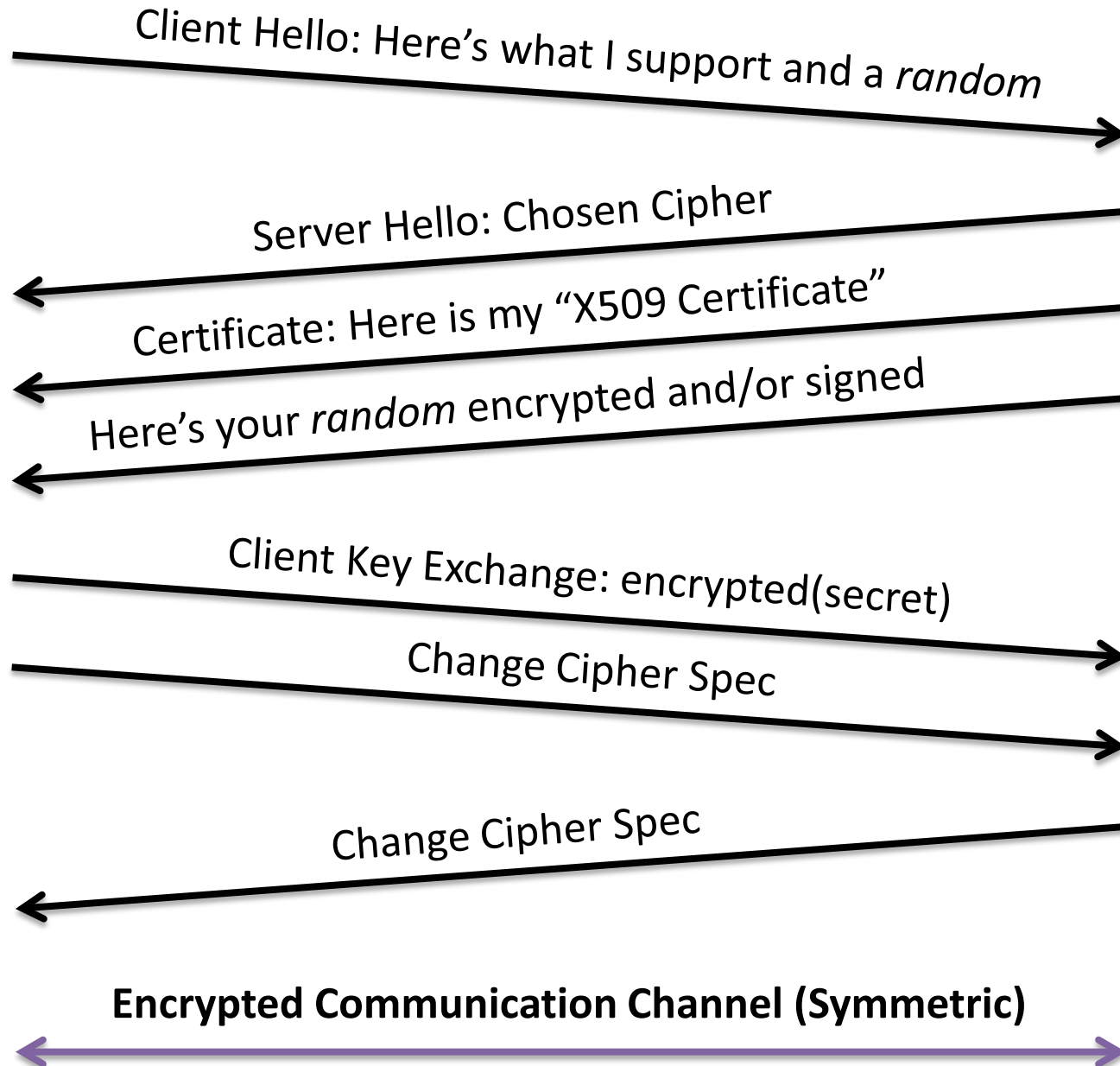
Client

Server



Client

Server



Cipher Suites

DHE - RSA - AES256 - SHA

Ephemeral
Key Exchange



```
graph BT; A[Ephemeral Key Exchange] --> C[DHE - RSA - AES256 - SHA]; B[Key Exchange] --> C; D[Data Transfer Cipher] --> C; E[Message Digest] --> C;
```

Key Exchange

Data Transfer
Cipher

Message Digest

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