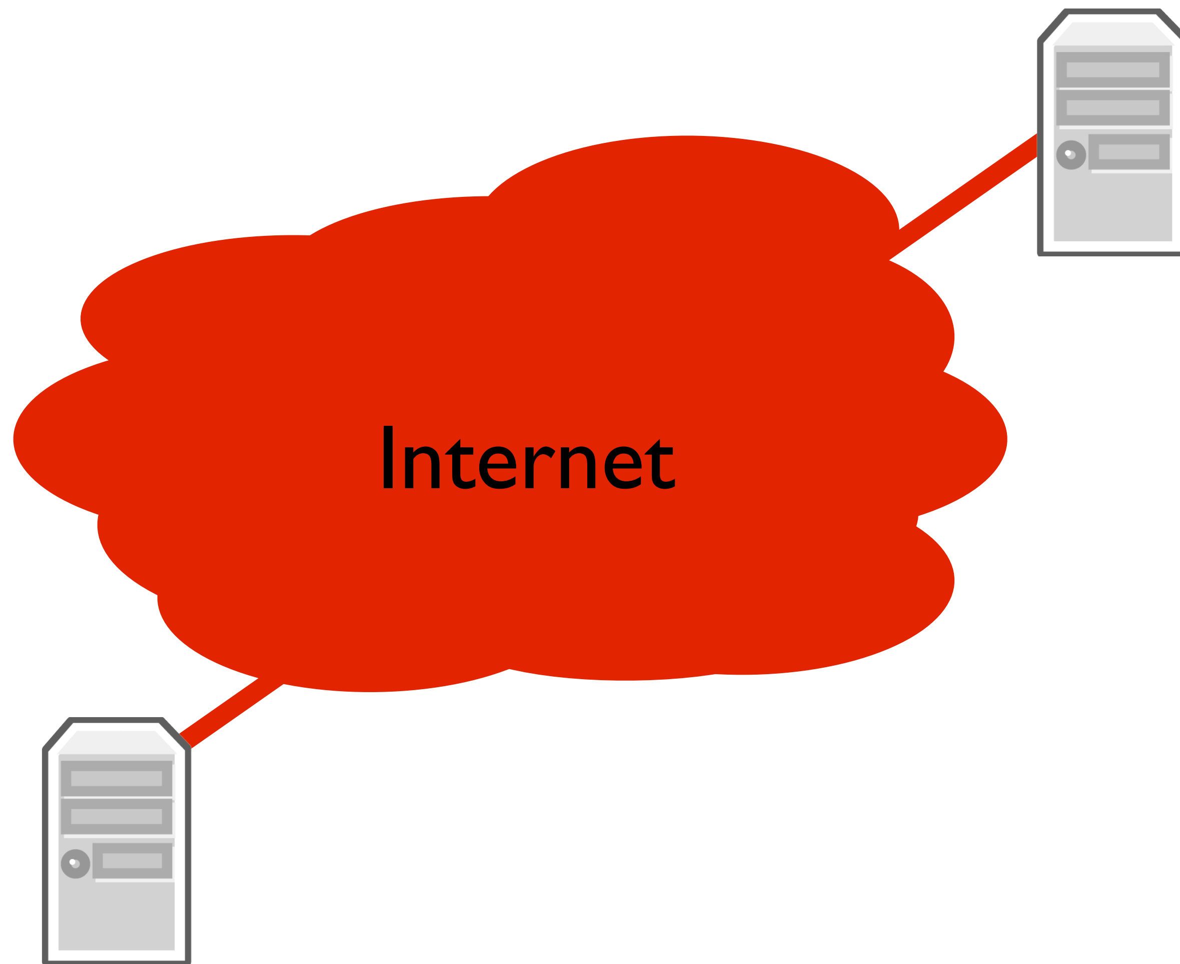
Unit 8 Review

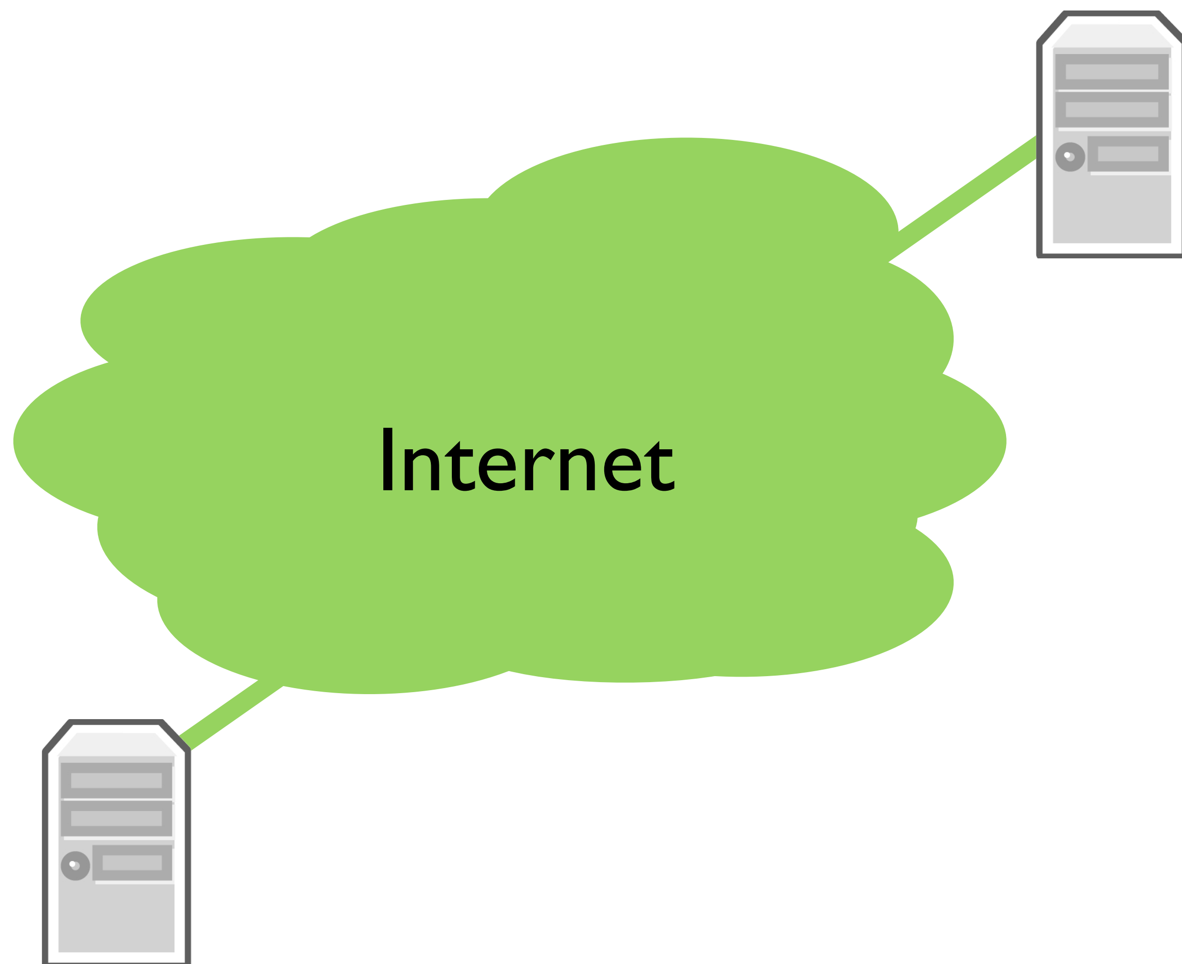
Secure your network!

Basic Problem



- To first approximation, attackers control the network
  - Can snoop, replay, suppress, send
- How do we defend against this?
  - Communicate securely despite insecure networks -- *cryptography*
  - Secure small parts of network despite insecurity of wider network
  - Design systems to scale well in response to attacks
- Two approaches: cryptography and scalable system design

Cryptography



- A set of mathematical principles and ideas for securing communication
- Be careful: easy to mess up!
  - Often misused!
  - Need to understand what it guarantees and what it doesn't
- How cryptography helps
  - Confidentiality: we can communicate privately (encryption)
  - Integrity: protect from tampering (hashes, signatures, MACs)
  - Authenticity: you are whom you say you are (certificates, MACs, signatures)

Confidentiality

- I want to tell you something secretly, so no-one else knows what I said
  - “My credit card number is....”
- Perfect confidentiality: one-time pad
  - You and I share a perfectly random *key* of zeroes and ones, K, nobody else has it
  - I XOR my message M with K, producing C, send C to you ($C = M \oplus K$)
  - You XOR C with K, you have reconstructed M ($M = C \oplus K$)
- Advantages: informationally theoretic secure and fast
  - Given any C, any M is equally likely
- Disadvantage: need a K as long as all data I might ever send

- Ways to exchange a small K such that there are 2^k possible Cs

Integrity

- I want to make sure you received my message unchanged/untampered
  - "I recalculated his grade and it is a C...", program to run on your computer
- I want to make sure you sent the message
  - Nick says: "I said it was alright if he handed in the assignment late"
- Cryptographic hash: $H(M)$
  - Turn arbitrary length input into fixed length hash
  - Collision-resistance: given $x \neq y$, intractable to find $H(x) = H(y)$
- Message authentication code: $MAC(M,K)$
  - Use key K to generate $MAC(M,K)$, use K to check $MAC(M,K)$
  - Intractable to generate $MAC(M,K)$ unless you have K

Authenticity

- I want to be sure you are whom you say you are
  - "This is the provost... I need your credit card number."
  - "You should send all of my traffic through this third party"
- We've exchanged a secret *K* beforehand: MAC("This is the...", K)
- If we haven't: chain of of trust
  - We can trust Verisign by design (root of trust)
  - Verisign says "here's a secret for Stanford"
  - Stanford says "here's a secret for the provost"

Symmetric Encryption

- Both parties, Alice and Bob, share a secret key K
- Given a message M and a key K
  - M is known as the plaintext or cleartext, what we're trying to keep secret
  - Encrypt: $E(K, M) \rightarrow C$, C is known as the ciphertext
  - Decrypt: $D(K, C) \rightarrow M$
  - Attacker cannot derive M from C without K (or trying every K...)

- E and D take the same key K, hence the name "symmetric" encryption
- Examples: AES, Blowfish, DES, RC4

One-Time Pad

- Generate a perfectly random stream of bits K
- $E(K, M) = M \oplus K$
- $D(K, C) = C \oplus K \quad (K \oplus K == 0)$
- "Perfect" secrecy
  - Informationally-theoretic secure: given C but not K, M could be anything!
  - Fast: XORing is cheap and fast
- Totally impractical
  - Need a very big key, same size as all data

Idea: Computational Security

- Distribute small *K* (e.g., 128 bits, 256 bits) securely
- Use *K* to encrypt much larger *M*
- Given C = *E(K, M)*, may be only one possible *M*
  - If *M* has redundancy
- But believed computationally intractable to find
  - Could try all possible *K*, but 2^{128} is a lot of work!

Security (1983)

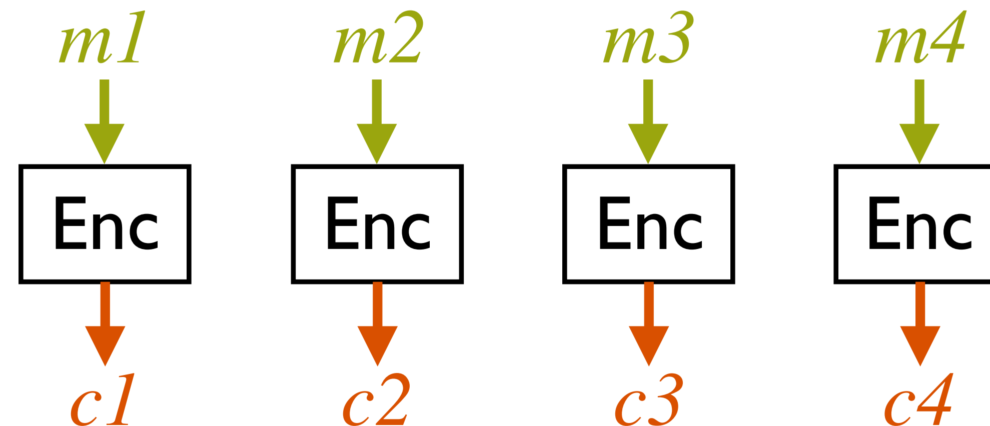
- Shafi Goldwasser and Silvio Micali, 2013 Turing Award
  - "By formalizing the concept that cryptographic security had to be computational rather than absolute, they created mathematical structures that turned cryptography from an art into a science."
- Define security in terms of computational complexity: e.g., it would take 1,000,000 computers 1,000,000 years to read your secret message

Ciphers

- Stream ciphers: pseudo-random pad
  - Generate a pseudo-random sequence of bits based on key
  - Encrypt/decrypt by XORing like one-time pad
  - BUT NOT A ONE-TIME PAD! Immediately mistrust anyone who says so!
  - Have run into many problems in practice, so I'd recommend avoiding them
    - Example: 802.11 WEP shown broken in ~2001, replaced by WPA in 2003, WPA2 in 2004
- Block ciphers
  - Operate on fixed sized blocks (64 bits, 128 bits, etc.)
  - Maps plaintext blocks to ciphertext blocks
  - Today, should use generally AES: many other algorithms

Using Block Ciphers

- Messages are typically longer than one block
- ECB (electronic code book) mode
  - Break plaintext into blocks, encrypt each block separately



  - Attacker can't decrypt any of the blocks; message secure
  - Can re-use keys, every block encrypted with cipher will be secure

WRONG!

- Attacker will learn of repeated plaintext blocks
  - If transmitting a sparse file, will know where non-zero regions lie
- Example: military instructions
  - Most days, send encrypted "nothing to report"
  - One day, send "attack today", attacker will know plans are being made

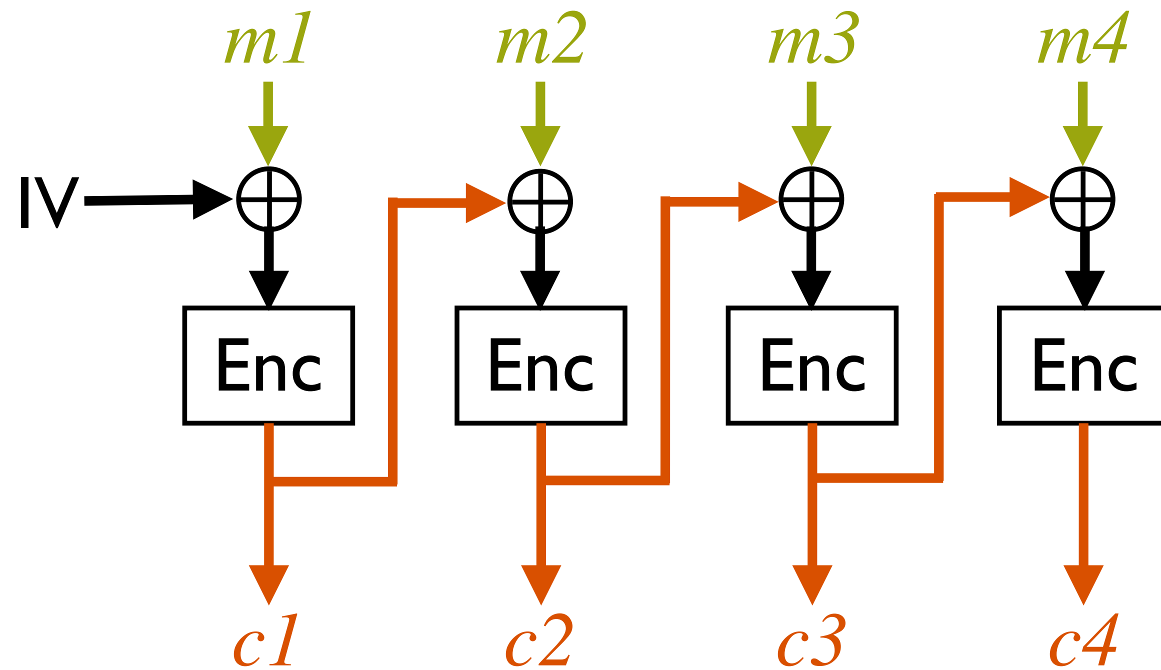
source

ECB

desired output

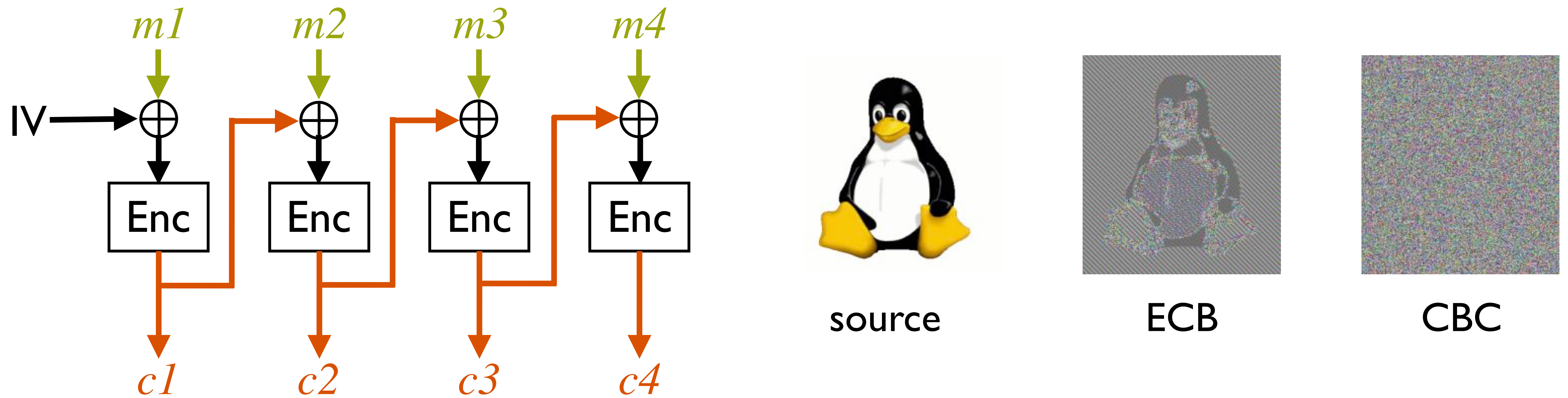
http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC) Mode



- Choose initialization vector (IV) for each message, can be publicly known
  - Can be 0 iff key only ever used to encrypt one message
  - Choose randomly for each message if key re-used
- $C_1 = E(K, M_1 \oplus IV), C_i = E(K, M_i \oplus C_{i-1})$

Cipher Block Chaining (CBC) Mode



- Choose initialization vector (IV) for each message, can be publicly known
  - Can be 0 iff key only ever used to encrypt one message
  - Choose randomly for each message if key re-used
- $C_1 = E(K, M_1 \oplus IV), C_i = E(K, M_i \oplus C_{i-1})$

Many Other Modes!

- Cipher Feedback (CFB) mode: $C_i = M_i \oplus E(K, C_{i-1})$
  - Useful for message that are not multiple of block size
- Output Feedback (OFB) mode
  - Repeatedly encrypt IV & use result like stream cipher
- Counter (CTR) mode: $C_i = M_i \oplus E(K, i)$
  - Useful if you want to encrypt in parallel, also means encryption same as decryption (IoT)

- Quiz: given a shared secret key, can you transmit files secured from adversaries over a network solely by encrypting them in CBC mode?

Secrecy Is Not Enough

- Encryption protects someone from reading plaintext
- An adversary can still modify messages
  - Flip a bit in the plaintext
  - Lead you to accept garbage data
- Integrity: protecting messages from tampering and modification
  - Node actually advertised that routing vector
  - Person actually made that bid
  - Endpoint actually sent that message to terminate the connection
- Confidentiality without integrity is rare (and a sign of a poor design), while integrity without confidentiality is common
  - Exception: encrypted storage (not on a network, no MitM)

Two Integrity Examples

- Cryptographic hashes
  - Way to verify that data has not been modified
  - Requires no secrets: anyone can generate one
  - Useful in data storage
- Message authentication codes (MACs)
  - Way to verify that data has not been modified
  - Also verifies generator has secret key: authenticity
  - Useful in networks

Cryptographic Hash

- Hash: computed fixed length output from arbitrary length input
  - Typical sizes 160-412 bits
  - Very cheap to compute, faster than network
- *Cryptographic* hash: also collision-resistant
  - Intractable to find $x \neq y$ such that $H(x) = H(y)$
  - Of course, many such collisions exist: ($2^{(2^{30} - 256)}$ GB blocks have same 256-bit hash)
  - But no-one has been able to find one, even after analyzing the algorithm for years
- Use SHA-256 or SHA-512 today (SHA-2)
  - Historically, most popular hash was SHA-1, but it's nearly broken
  - October 2, 2012: Keccak algorithm chosen by NIST for SHA-3
    - Goal: alternative, dissimilar hash function to SHA-2

Designed by NSA

Designed by Bertoni et al.

HMAC

- Anyone can generate a cryptographic hash: MACs want to also provide assurance generator has shared secret
- Simple approach: generate MAC from hash and secret,
  - HMAC(K,M) = SHA-2(K, M)
  - Send {M, HMAC(K,M)}
- Simple: we have a cryptographically strong MAC!

WRONG

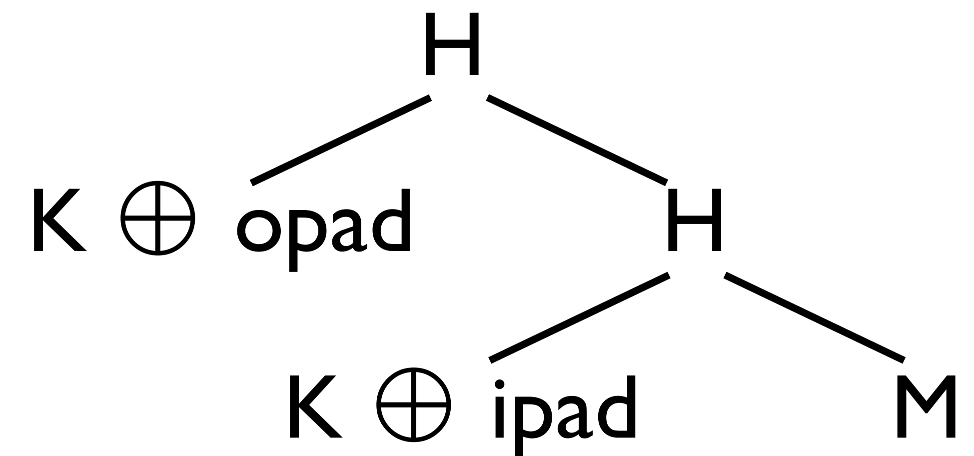
- Anyo[...]o provide assu[...]
- Simp[...]
  - HI[...]
  - Se[...]
- Simple: we have a cryptographically strong MAC!

If I have {M, HMAC(K,M)},
I can generate {M', HMAC(K,M')},
where M' has data appended to M

HMAC, Revisited

- HMAC(K,M) = H(K ⊕ opad, H(K ⊕ ipad, M))
  - H is a cryptographic hash such as SHA-3
  - ipad is 0x36 repeated 64 times, opad 0x5c repeated 64 times

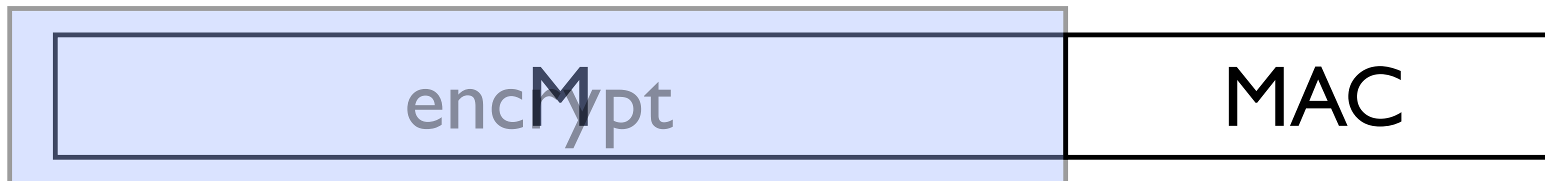
- Why does previous attack not work?



Encryption and MACs

- Should I encrypt the MAC, or MAC the encrypted data?

or



- Encrypting the MAC is not always secure
- MACing encrypted data is always secure

Popular Public Key Algorithms

- Encryption: RSA, Rabin, ElGamal
- Signature: RSA, Rabin. ElGamal, Schnorr, DSA, ...
- Warning: message padding critically important
  - Basic idea behind RSA encryption is simple (modular exponentiation of large integers)
  - But simple transformations of message to numbers is not secure
- Many keys support both signing and encryption
  - But they use different algorithms
  - Common error: Sign by "encrypting" with private key

Confidentiality

- Three (randomized) algorithms
  - Generate: $G(1^k) \rightarrow K, K^{-1}$ (randomized)
  - Encrypt: $E(K,m) \rightarrow \{m\}_K$ (randomized)
  - Decrypt: $D(K^{-1}, \{m\}_K) \rightarrow m$
- Provides confidentiality: can't derive m from $\{m\}_K$ without K^{-1}
  - K can be made public: can't derive K^{-1} from K
  - Everyone can share the same K ("public" key)
- Encrypt must be randomized
  - Same plaintext sent multiple times must generate different ciphertexts
  - Otherwise can easily guess plaintext for small message space ("yes", "no")

Integrity: Signatures

- Three (randomized) algorithms
  - Generate: $G(1^k) \rightarrow K, K^{-1}$ (randomized)
  - Sign: $S(K^{-1}, m) \rightarrow \{m\}_{K^{-1}}$ (can be randomized)
  - Verify: $V(K, \{m\}_{K^{-1}}, m) \rightarrow$ {yes, no}
- Provides integrity like a MAC
  - Cannot produce valid $(\{m\}_{K^{-1}}, m)$ pair without K^{-1}
  - But only need K to verify, cannot derive K^{-1} from K
  - So K can be publicly known

The Catch

- Cost of public key algorithms is significant

Algorithm	Encrypt	Decrypt	Sign	Verify
RSA 1024	0.08ms	1.46ms	1.48ms	0.07ms
RSA 2048	0.16ms	6.08ms	6.05ms	0.16ms
DSA 1024			0.45ms	0.83ms
LUC 2048	0.18ms	9.89ms	9.92ms	0.18ms
DLIES 2048	4.11ms	3.86ms		

- In contrast, symmetric algorithms can operate at line speed

http://www.cryptopp.com/benchmarks.html

Problem

- Want to communicate securely with a server, e.g. www.amazon.com
  - Using public key cryptography, given a public key K, we can verify that another program has the associated private key K^{-1}
    - Verify: $V(K, \{m\}_{K^{-1}}, m) \rightarrow$ {yes, no}
  - Use public key cryptography to exchange symmetric keys
- Problem: key management
  - How do we get the server's public key?
  - How can we be sure it's the server's public key?

Certificate

- Certificate: document signed by a private key K_1^{-1} that binds a public key K_2 to an identity/name N
  - "The public key of www.ebay.com is ...."
  - "The public key of axess.stanford.edu is..."
- If we trust the signing party and know their public key, we can use K_2 when communicating with N

Bad Certificate!

This Connection is Untrusted

You have asked Firefox to connect securely to , but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

▶ **Technical Details**

▶ **I Understand the Risks**

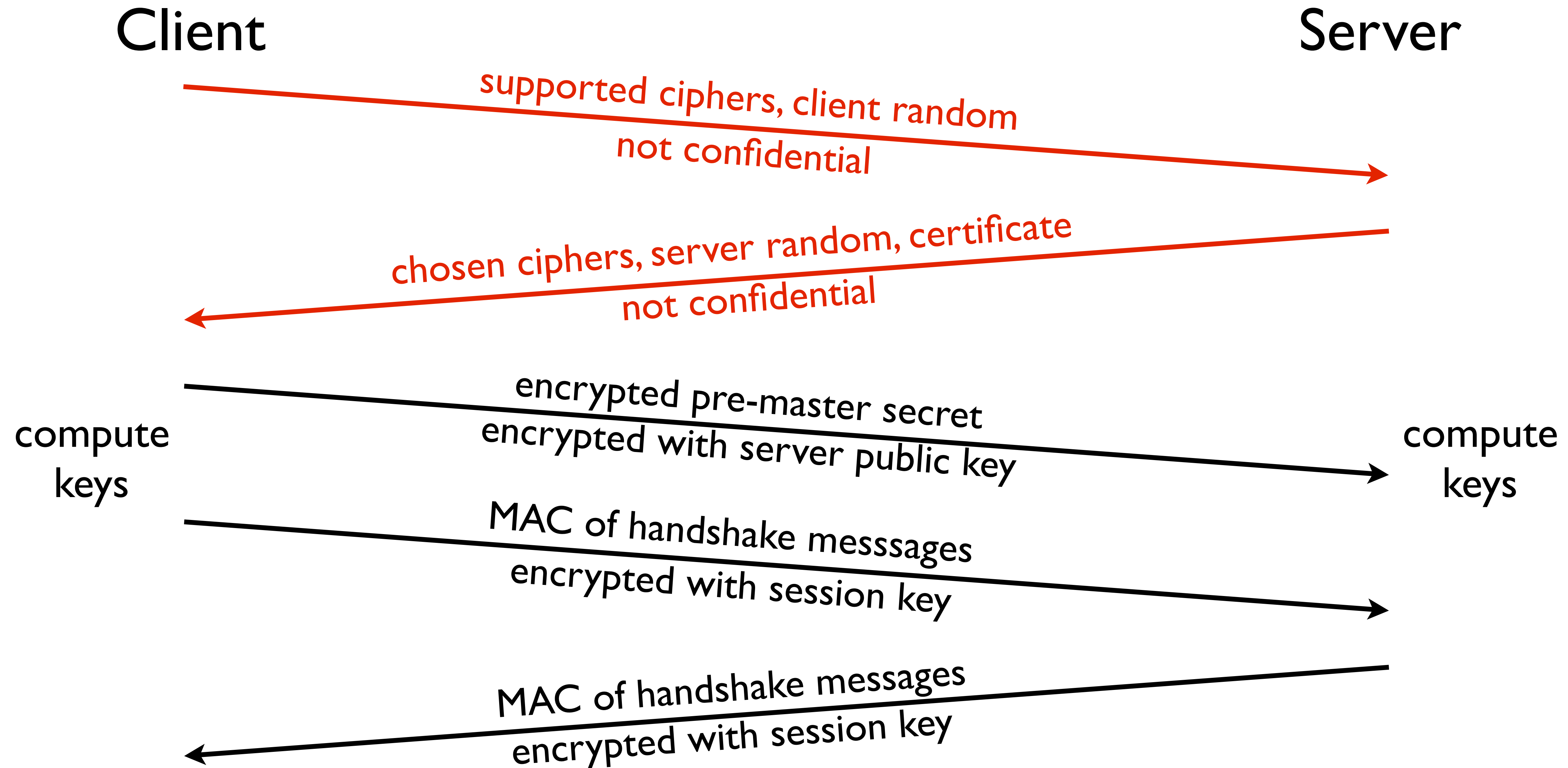
Bootstrapping Trust

- "Everyone" trusts a few signing authorities and knows their public keys
  - Hard-baked into your browser or OS
  - Examples: Verisign, Microsoft, Google
- Root authorities sign certificates (e.g., for Stanford)
- Can then use those certificates to sign further certificates
  - "Chain of trust" - GeoTrust to Google to www.google.com
- Certificate only says that someone testifies that a host has this key! Have to trust every step along chain
- This is how TLS/HTTPS works today (the padlock in your browser bar)

Hybrid Schemes

- Use public key to encrypt symmetric key
- Negotiate secret session key
  - Use public key crypto to establish 4 symmetric keys
  - Client sends server $\{\{m_1\}_{K1}, HMAC(K_2, \{m1\}_{K1})\}$
  - Server sends client $\{\{m_2\}_{K3}, HMAC(K_4, \{m_2\}_{K3})\}$
- Common pitfall: signing underspecified messages
  - E.g., always specify intended recipient in signed messages
  - Should also specify expiration, or better yet fresh data
  - Otherwise like signing a blank check to anyone for all time...

TLS Handshake



Establishing Session Keys

- Both client and server add randomness
- Client sends "pre-master secret" encrypted with server's public key
- Use randomness and pre-master secret to generate "master secret"
- Use master secret and random numbers to generate session keys
  - Client to server write (encryption)
  - Client to server MAC
  - Server to client write (encryption)
  - Server to client MAC
  - Client initialization vector
  - Server initialization vector
- Support for resuming sessions with same master secret (but new keys)

Session Key Details