Secure Communication with an Insecure Internet Infrastructure

Internet Design Decisions and Security

- Origin as a small and cooperative network
  (=> largely trusted infrastructure)
- Global Addressing
  (=> every sociopath is your next-door neighbor*)
- Connection-less datagram service
  (=> can't verify source, hard to protect bandwidth)

* Dan Geer

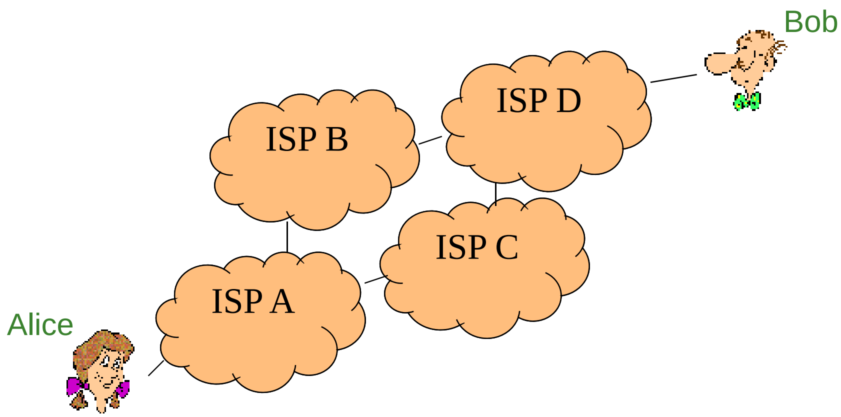
Internet Design Decisions and Security

- Anyone can connect
  (=> ANYONE can connect)
- Millions of hosts run nearly identical software
  (=> single exploit can create epidemic)
- Most Internet users know about as much as Senator Stevens aka “the tubes guy”
  (=> God help us all…)

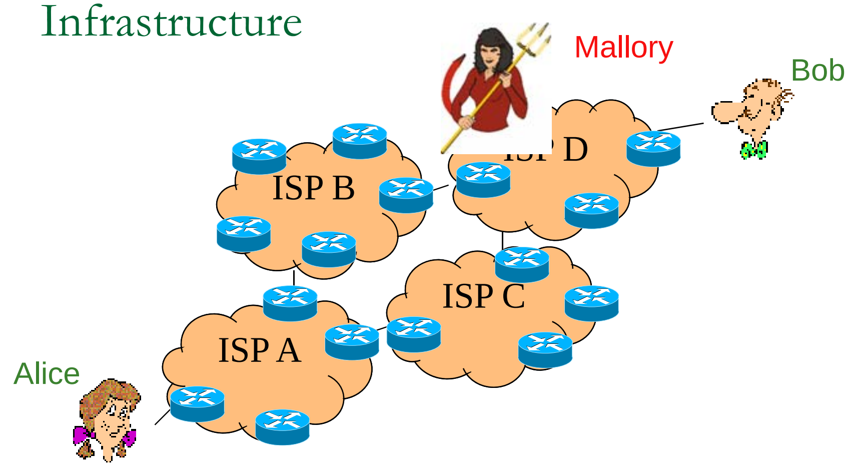
Our “Narrow” Focus

- Yes:
  - Protecting network resources and limiting connectivity (Last time)
  - Creating a “secure channel” for communication (today)
- No:
  - Preventing software vulnerabilities & malware, or “social engineering”.

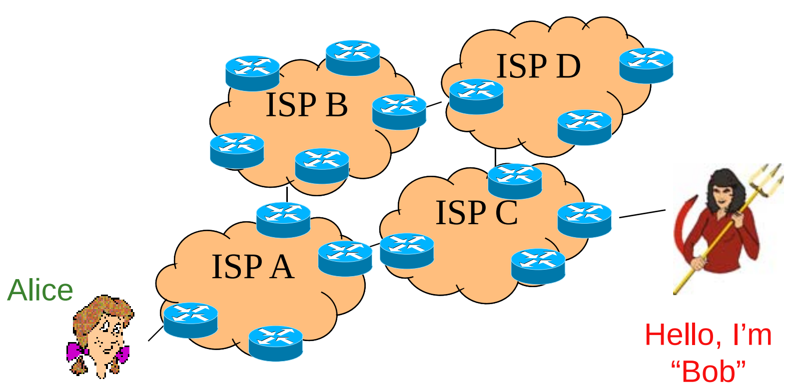
Secure Communication with an Untrusted Infrastructure



Secure Communication with an Untrusted Infrastructure



Secure Communication with an Untrusted Infrastructure



What do we need for a secure communication channel?

- Authentication (Who am I talking to?)
- Confidentiality (Is my data hidden?)
- Integrity (Has my data been modified?)
- Availability (Can I reach the destination?)

What is cryptography?

"cryptography is about communication in the presence of adversaries."

- Ron Rivest

“cryptography is using math and other crazy tricks to approximate magic”

- Unknown 441 TA

What is cryptography?

Tools to help us build secure communication channels that provide:

1) Authentication
2) Integrity
3) Confidentiality

Cryptography As a Tool

- Using cryptography securely is not simple
- Designing cryptographic schemes correctly is near impossible.

Today we want to give you an idea of what can be done with cryptography.

Take a security course if you think you may use it in the future (e.g. 18-487)

The Great Divide

	Symmetric Crypto (Private key) (E.g., AES)	Asymmetric Crypto (Public key) (E.g., RSA)
Shared secret between parties?	Yes	No
Speed of crypto operations	Fast	Slow

Symmetric Key: Confidentiality

Motivating Example:

You and a friend share a key K of L random bits, and want to secretly share message M also L bits long.

Scheme:

You send her the *xor(M,K)* and then she "decrypts" using *xor(M,K)* again.

1) Do you get the right message to your friend?
2) Can an adversary recover the message M?
3) Can adversary recover the key K?

Symmetric Key: Confidentiality

- One-time Pad (OTP) is secure but usually impactical
  - Key is as long at the message
  - Keys cannot be reused (why?)

In practice, two types of ciphers are used that require constant length keys:

Stream Ciphers:

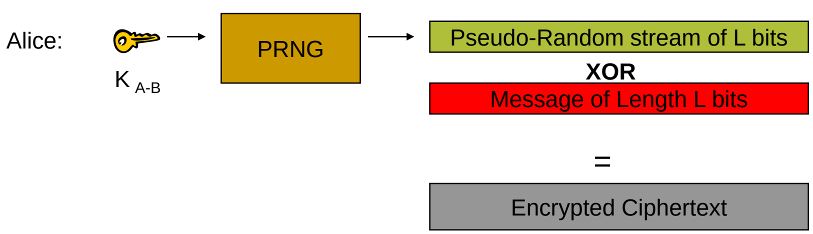
Ex: RC4, A5

Block Ciphers:

Ex: DES, AES, Blowfish

Symmetric Key: Confidentiality

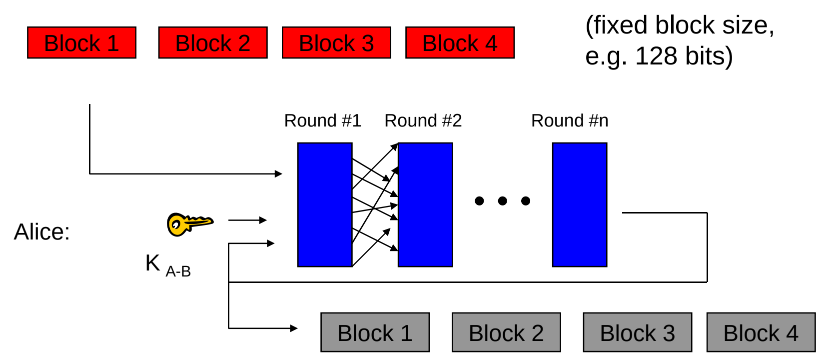
- Stream Ciphers (ex: RC4)



Bob uses K_{A-B} as PRNG seed, and XORs encrypted text to get the message back (just like OTP).

Symmetric Key: Confidentiality

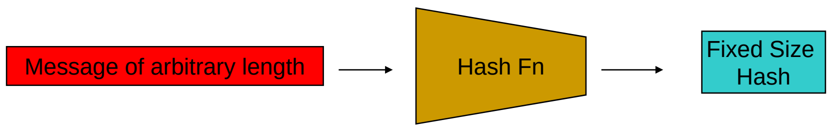
- Block Ciphers (ex: AES)



Bob breaks the ciphertext into blocks, feeds it through decryption engine using K_{A-B} to recover the message.

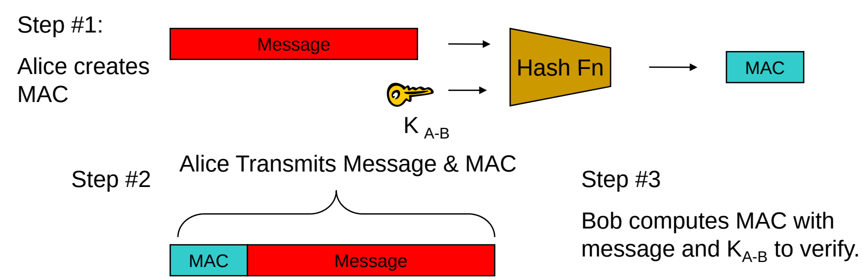
Cryptographic Hash Functions

- Consistent
  - hash(X) always yields same result
- One-way
  - given Y, can't find X s.t. hash(X) = Y
- Collision resistant
  - given hash(W) = Z, can't find X such that hash(X) = Z



Symmetric Key: Integrity

- Hash Message Authentication Code (HMAC)

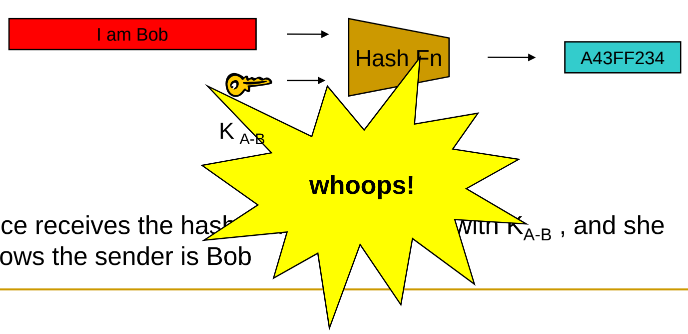


Why is this secure?
How do properties of a hash function help us?

Symmetric Key: Authentication

- You already know how to do this!

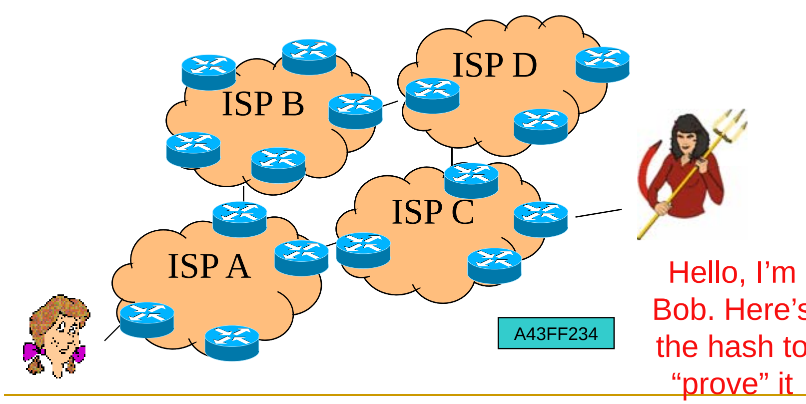
  (hint: think about how we showed integrity)



Alice receives the hash ... with K_{A-B}, and she knows the sender is Bob

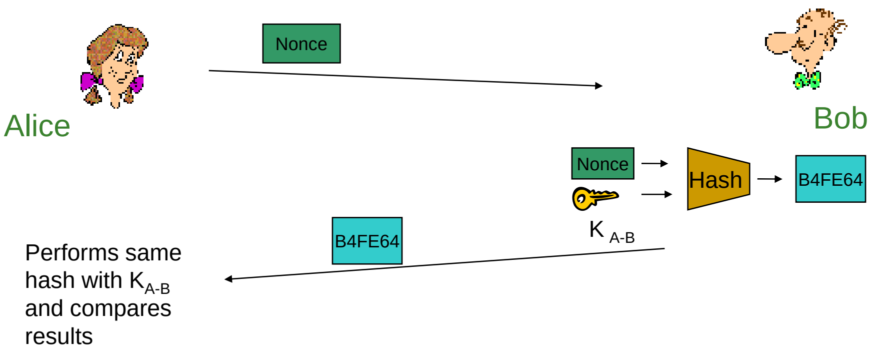
Symmetric Key: Authentication

What if Mallory overhears the hash sent by Bob, and then "replays" it later?



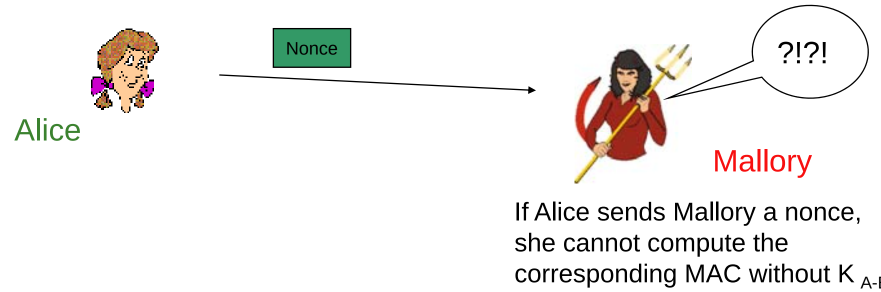
Symmetric Key: Authentication

- A "Nonce"
  - A random bitstring used only once. Alice sends nonce to Bob as a "challenge". Bob Replies with "fresh" MAC result.



Symmetric Key: Authentication

- A "Nonce"
  - A random bitstring used only once. Alice sends nonce to Bob as a "challenge". Bob Replies with "fresh" MAC result.



Symmetric Key Crypto Review

- Confidentiality: Stream & Block Ciphers
- Integrity: HMAC
- Authentication: HMAC and Nonce

Questions??

Are we done? Not Really:

1) Number of keys scales as $O(n^2)$

2) How to securely share keys in the first place?

Diffie-Hellman key exchange

- An early (1976) way to create a shared secret.
- Everyone knows a prime, p, and a generator, g.
- Alice and Bob want to share a secret, but only have internet to communicate over.

DH key exchange

Everyone: large prime p and generator g

Create secret: a

Alice

Send Bob: $g^a \bmod p$ →

Create secret: b

Bob

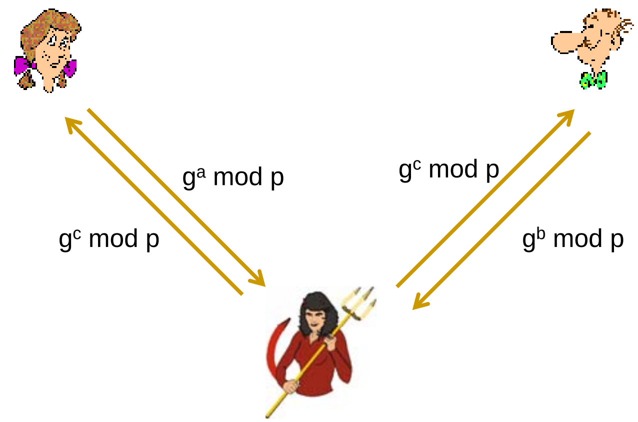
← Send Alice: $g^b \bmod p$

Compute: $(g^b \bmod p)^a$

Compute: $(g^a \bmod p)^b$

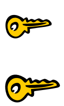
Voila: They both know g^{ab} which is secret!

DH key exchange & Man-In-The-Middle



Asymmetric Key Crypto:

- Instead of shared keys, each person has a "key pair"



K_B Bob's <u>public</u> key

K_B^{-1} Bob's <u>private</u> key

- The keys are inverses, so: $K_B^{-1}(K_B(m)) = m$

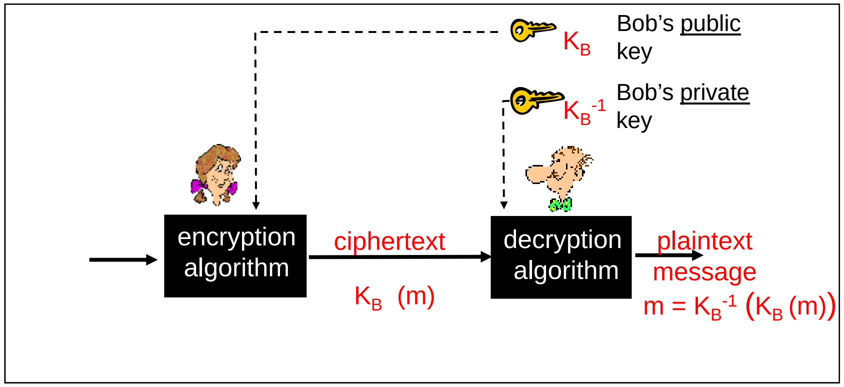
Asymmetric Key Crypto:

- It is believed to be computationally unfeasible to derive K_B^{-1} from K_B or to find any way to get M from $K_B(M)$ other than using K_B^{-1} .

=> K_B can safely be made public.

Note: We will not explain the computation that $K_B(m)$ entails, but rather treat these functions as black boxes with the desired properties.

Asymmetric Key: Confidentiality



Asymmetric Key: Sign & Verify

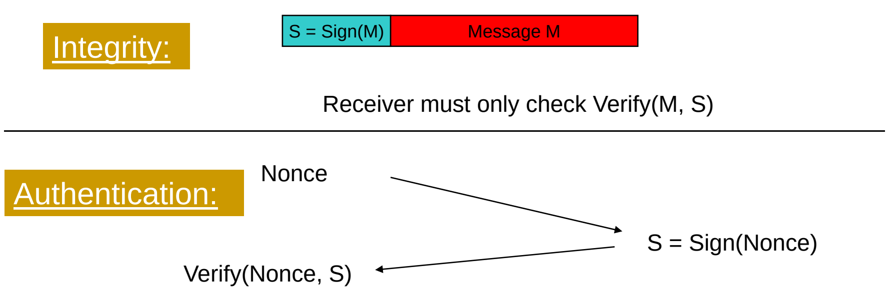
- If we are given a message M, and a value S such that $K_B(S) = M$, what can we conclude?

- The message must be from Bob, because it must be the case that $S = K_B^{-1}(M)$, and only Bob has K_B^{-1}!

  - This gives us two primitives:
    - Sign (M) = $K_B^{-1}(M)$ = Signature S
    - Verify (S, M) = test($K_B(S)$ == M)

Asymmetric Key: Integrity & Authentication

- We can use Sign() and Verify() in a similar manner as our HMAC in symmetric schemes.



Asymmetric Key Review:

- Confidentiality: Encrypt with Public Key of Receiver
- Integrity: Sign message with private key of the sender
- Authentication: Entity being authenticated signs a nonce with private key, signature is then verified with the public key

But, these operations are computationally expensive*

One last "little detail"…

How do I get these keys in the first place??
Remember:

- Symmetric key primitives assumed Alice and Bob had already shared a key.
- Asymmetric key primitives assumed Alice knew Bob's public key.

This may work with friends, but when was the last time you saw Amazon.com walking down the street?

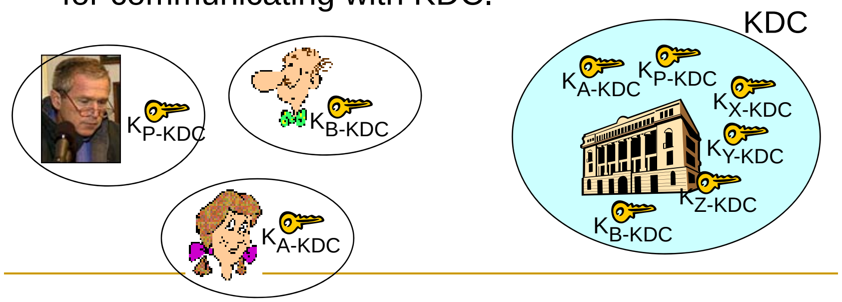
Symmetric Key Distribution

- How does Andrew do this?

Andrew Uses Kerberos, which relies on a Key Distribution Center (KDC) to establish shared symmetric keys.

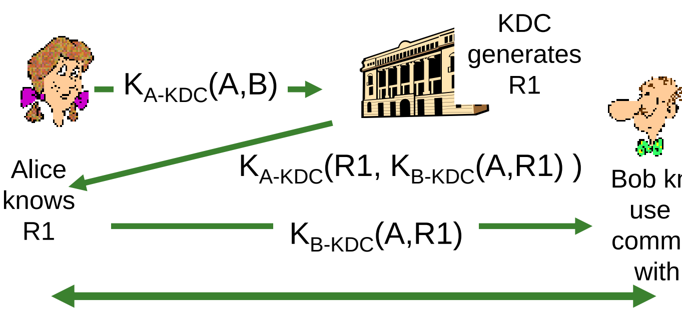
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key.
- KDC: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, K_{A-KDC} K_{B-KDC} , for communicating with KDC.



Key Distribution Center (KDC)

Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



Alice and Bob communicate: using R1 as *session key* for shared symmetric encryption

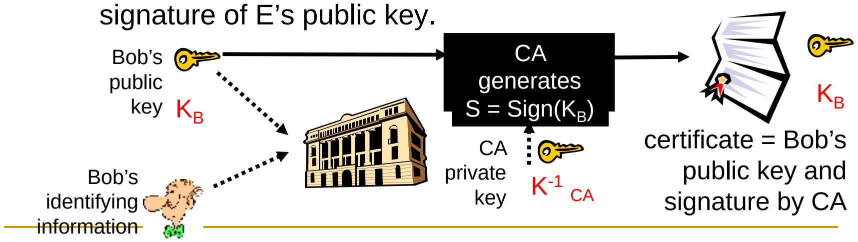
How Useful is a KDC?

- Must always be online to support secure communication
- KDC can expose our session keys to others!
- Centralized trust and point of failure.

In practice, the KDC model is mostly used within single organizations (e.g. Kerberos) but not more widely.

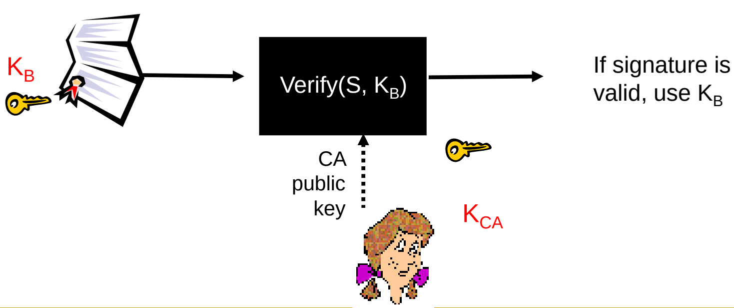
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- An entity E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - Certificate contains E's public key AND the CA's signature of E's public key.



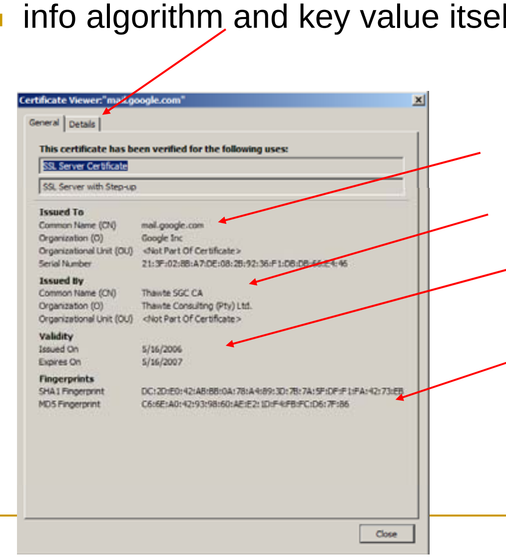
Certification Authorities

- When Alice wants Bob's public key:
  - Gets Bob's certificate (Bob or elsewhere).
  - Use CA's public key to verify the signature within Bob's certificate, then accepts public key



Certificate Contents

- info algorithm and key value itself (not shown)



- Cert owner
- Cert issuer
- Valid dates
- Fingerprint of signature

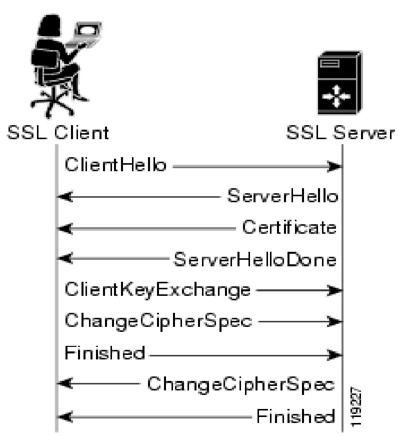
Which Authority Should You Trust?

- Today: many authorities
- What about a shared Public Key Infrastructure (PKI)?
  - A system in which “roots of trust” authoritatively bind public keys to real-world identities
  - So far it has not been very successful

Transport Layer Security (TLS) aka Secure Socket Layer (SSL)

- Used for protocols like HTTPS
- Special TLS socket layer between application and TCP (small changes to application).
- Handles confidentiality, integrity, and authentication.
- Uses “hybrid” cryptography.

Setup Channel with TLS “Handshake”



Handshake Steps:

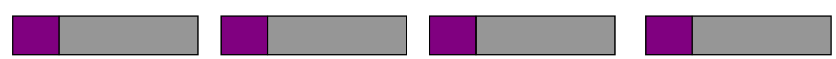
1) Client and server negotiate exact cryptographic protocols
2) Client validates public key certificate with CA public key.
3) Client encrypts secret random value with server’s key, and sends it as a challenge.
4) Server decrypts, proving it has the corresponding private key.
5) This value is used to derive symmetric session keys for encryption & MACs.

How TLS Handles Data

1) Data arrives as a stream from the application via the TLS Socket

2) The data is segmented by TLS into chunks

3) A session key is used to encrypt and MAC each chunk to form a TLS “record”, which includes a short header and data that is encrypted, as well as a MAC.



4) Records form a byte stream that is fed to a TCP socket for transmission.

What to take home?

- Internet design and growth => security challenges
- Symmetric (pre-shared key, fast) and asymmetric (key pairs, slow) primitives provide:
  - Confidentiality
  - Integrity
  - Authentication
- "Hybrid Encryption" leverages strengths of both.
- Great complexity exists in securely acquiring keys.
- Crypto is hard to get right, so use tools from others, don't design your own (e.g. TLS).

Resources

- Textbook: 8.1 – 8.3
- Wikipedia for overview of Symmetric/Asymmetric primitives and Hash functions.
- OpenSSL (www.openssl.org): top-rate open source code for SSL and primitive functions.
- "Handbook of Applied Cryptography" available free online: www.cacr.math.uwaterloo.ca/hac/