CSCI-1680
Security

Chen Avin

Based on lecture notes by Scott Shenker and Mike Freedman and Rodrigo Fonseca

Today's Lecture

- Classes of attacks
- Basic security requirements
- Simple cryptographic methods
- Cryptographic toolkit (Hash, Digital Signature, …)
- Certificate Authorities
- SSL / HTTPS

Basic Requirements for Secure Communication

- **Availability: Will the network deliver data?**
  - Infrastructure compromise, DDoS
- **Authentication: Who is this actor?**
  - Spoofing, phishing
- **Integrity: Do messages arrive in original form?**
- **Confidentiality: Can adversary read the data?**
  - Sniffing, man-in-the-middle
- **Provenance: Who is responsible for this data?**
  - Forging responses, denying responsibility
  - Not who sent the data, but who created it

Other Desirable Security Properties

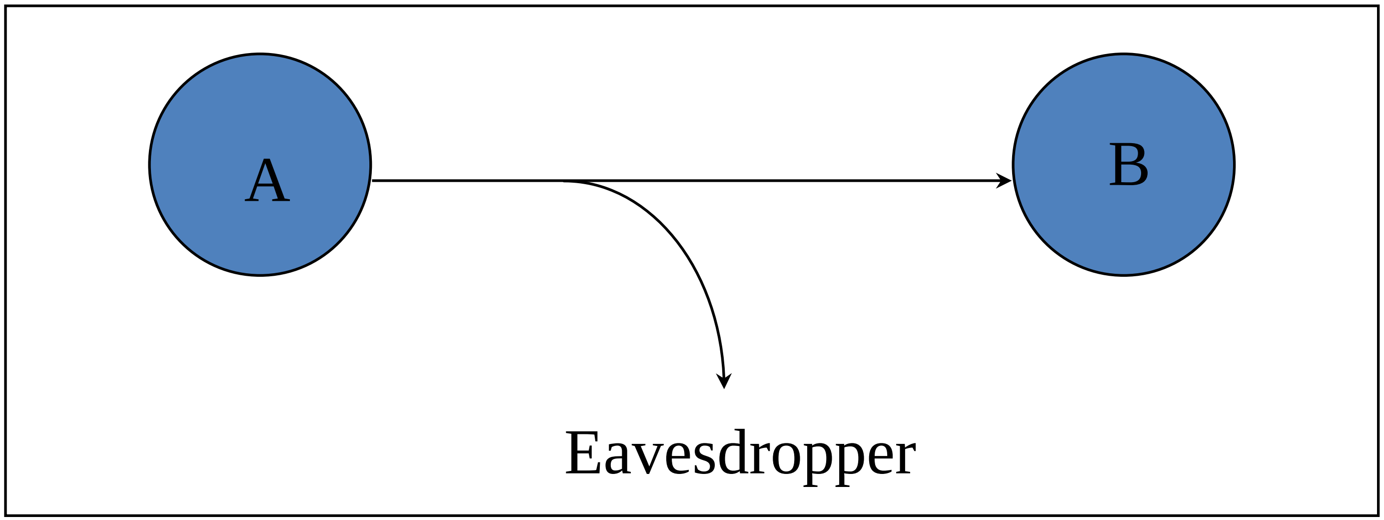
- **Authorization: is actor allowed to do this action?**
  - Access controls
- **Accountability/Attribution: who did this activity?**
- **Audit/Forensics: what occurred in the past?**
  - A broader notion of accountability/attribution
- **Appropriate use: is action consistent with policy?**
  - E.g., no spam; no games during business hours; etc.
- **Freedom from traffic analysis: can someone tell when I am sending and to whom?**
- **Anonymity: can someone tell I sent this packet?**

Internet’s Design: Insecure

- Designed for simplicity in a naïve era
- “On by default” design
- Readily available zombie machines
- Attacks look like normal traffic
- Internet’s federated operation obstructs cooperation for diagnosis/mitigation

Eavesdropping - Message Interception (Attack on Confidentiality)

- **Unauthorized access to information**
- **Packet sniffers and wiretappers**
- **Illicit copying of files and programs**



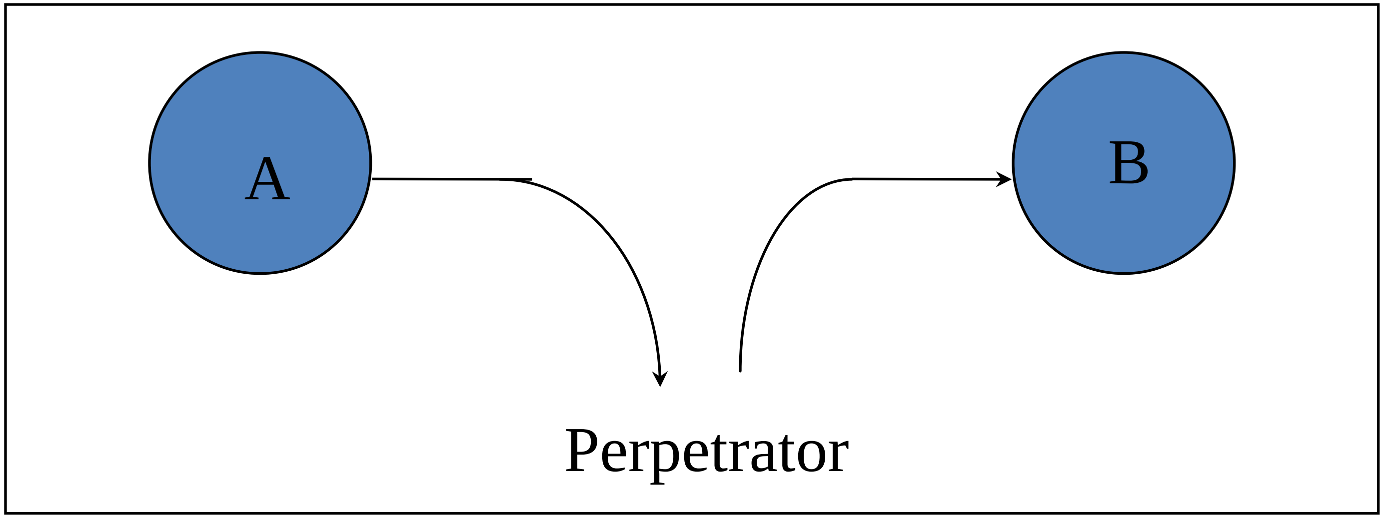
Eavesdropping Attack: Example

- **tcpdump with promiscuous network interface**
  - On a switched network, what can you see?

- **What might the following traffic types reveal about communications?**
  - DNS lookups (and replies)
  - IP packets without payloads (headers only)
  - Payloads

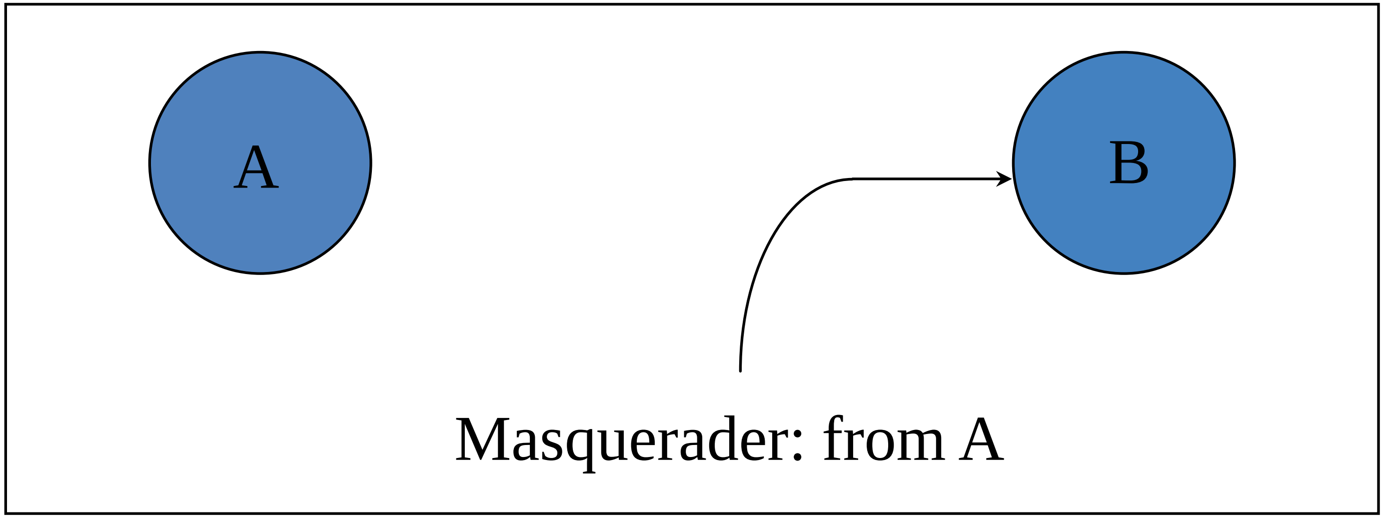
Integrity Attack - Tampering

- Stop the flow of the message
- Delay and optionally modify the message
- Release the message again



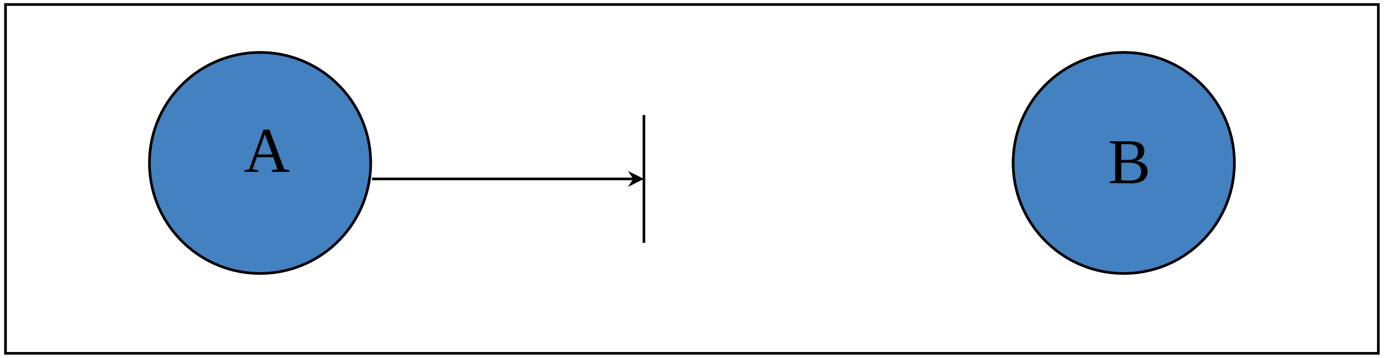
Authenticity Attack - Fabrication

- **Unauthorized assumption of other's identity**
- **Generate and distribute objects under this identity**



Attack on Availability

- **Destroy hardware (cutting fiber) or software**
- **Modify software in a subtle way**
- **Corrupt packets in transit**



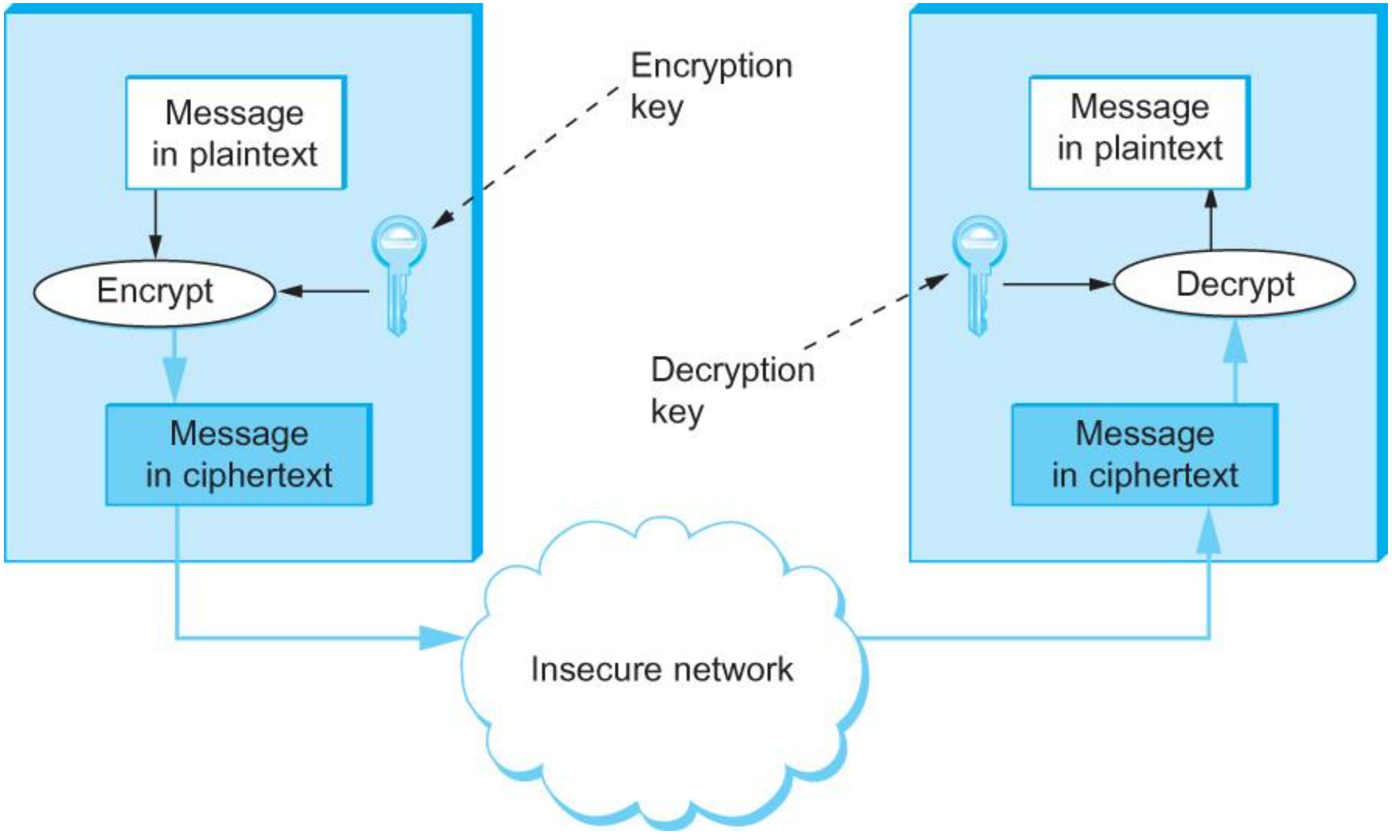
- **Blatant *denial of service* (DoS):**
  - Crashing the server
  - Overwhelm the server (use up its resource)

Basic Forms of Cryptography

Confidentiality through Cryptography

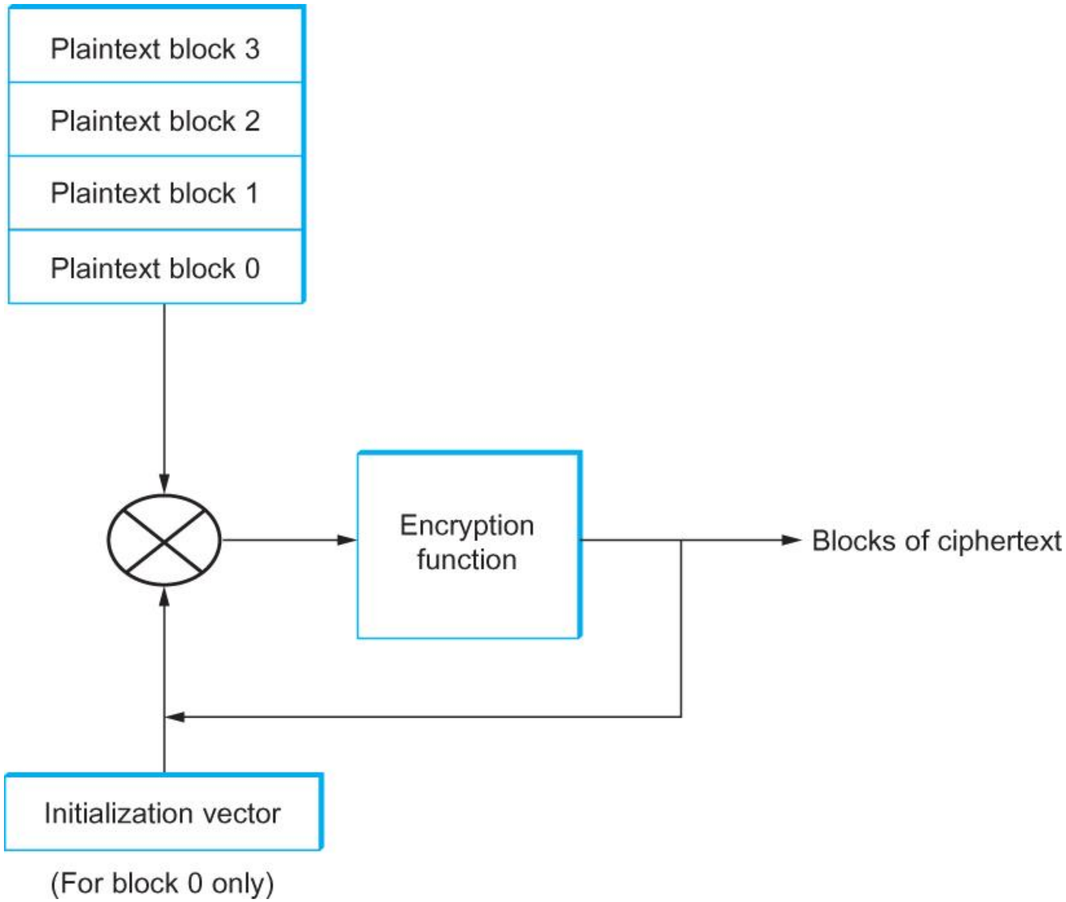
- **Cryptography:** *communication over insecure channel in the presence of adversaries*
- **Studied for thousands of years**
- **Central goal:** how to encode information so that an adversary can't extract it …but a friend can
- **General premise:** a ***key*** is required for decoding
  - Give it to friends, keep it away from attackers
- **Two different categories of encryption**
  - Symmetric: efficient, requires key distribution
  - Asymmetric (Public Key): computationally expensive, but no key distribution problem

Principles of Ciphers



- Known plaintext attack
- Ciphetext only attack
- Chosen plaintext attack

Block Ciphers



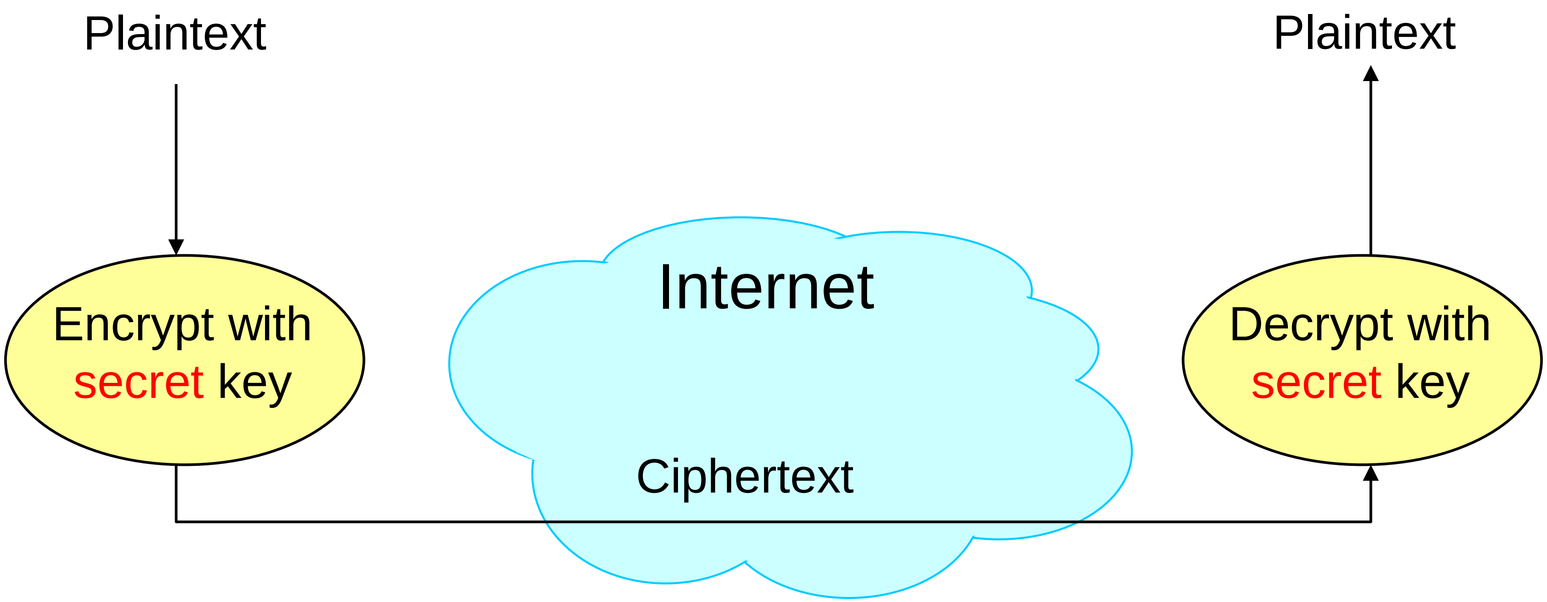
Cipher block chaining (CBC).

Symmetric Key Encryption

- **Same key for encryption and decryption**
  - Both sender and receiver know key
  - But adversary does not know key
- **For communication, problem is key distribution**
  - How do the parties (secretly) agree on the key?
- **What can you do with a huge key? One-time pad**
  - Huge key of random bits
- **To encrypt/decrypt: just XOR with the key!**
  - Provably secure! …. provided:
    - You never reuse the key … and it really is random/unpredictable
  - Spies actually use these

Using Symmetric Keys

- **Both the sender and the receiver use the same secret keys**

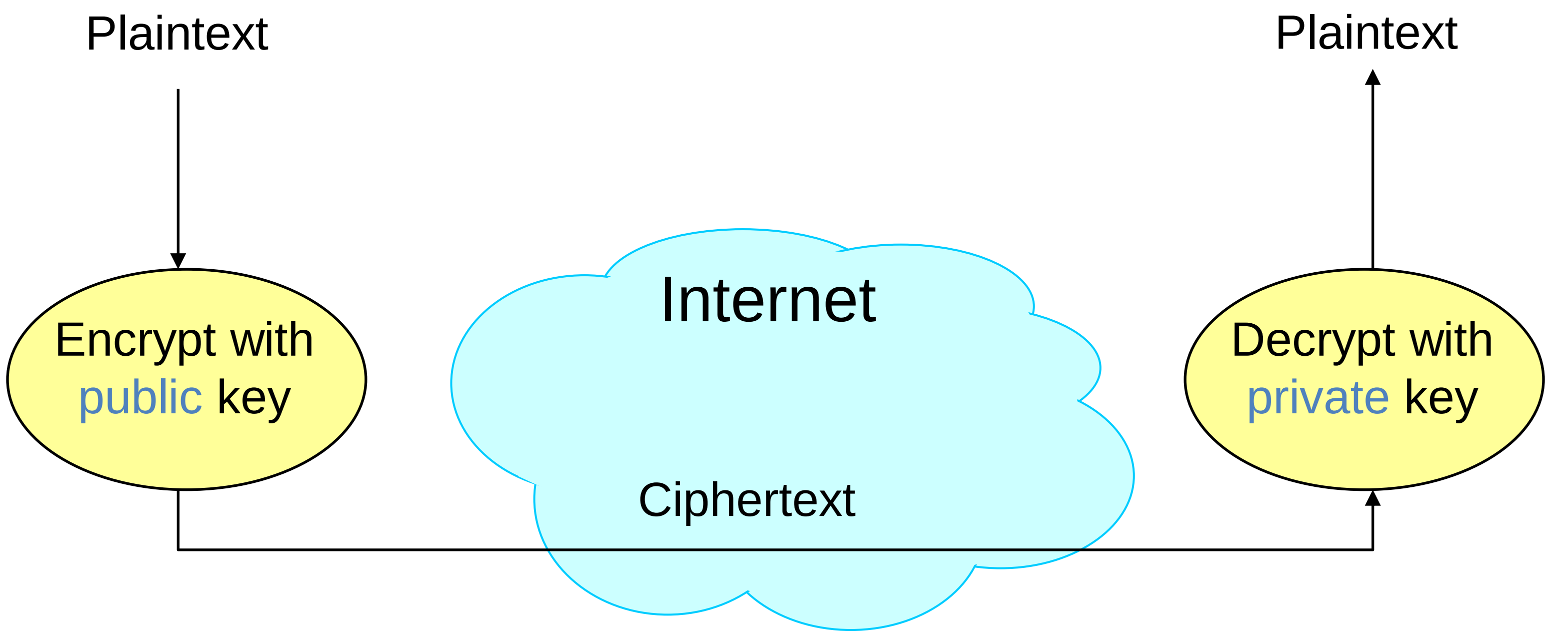


Asymmetric Encryption *(Public Key)*

- **Idea: use two *different* keys, one to encrypt (*e*) and one to decrypt (*d*)**
  - A key pair
- **Crucial property: knowing *e* does not give away *d***
- **Therefore *e* can be public: everyone knows it!**
- **If Alice wants to send to Bob, she fetches Bob's public key (say from Bob's home page) and encrypts with it**
  - Alice can't decrypt what she's sending to Bob …
  - … but then, neither can anyone else (except Bob)

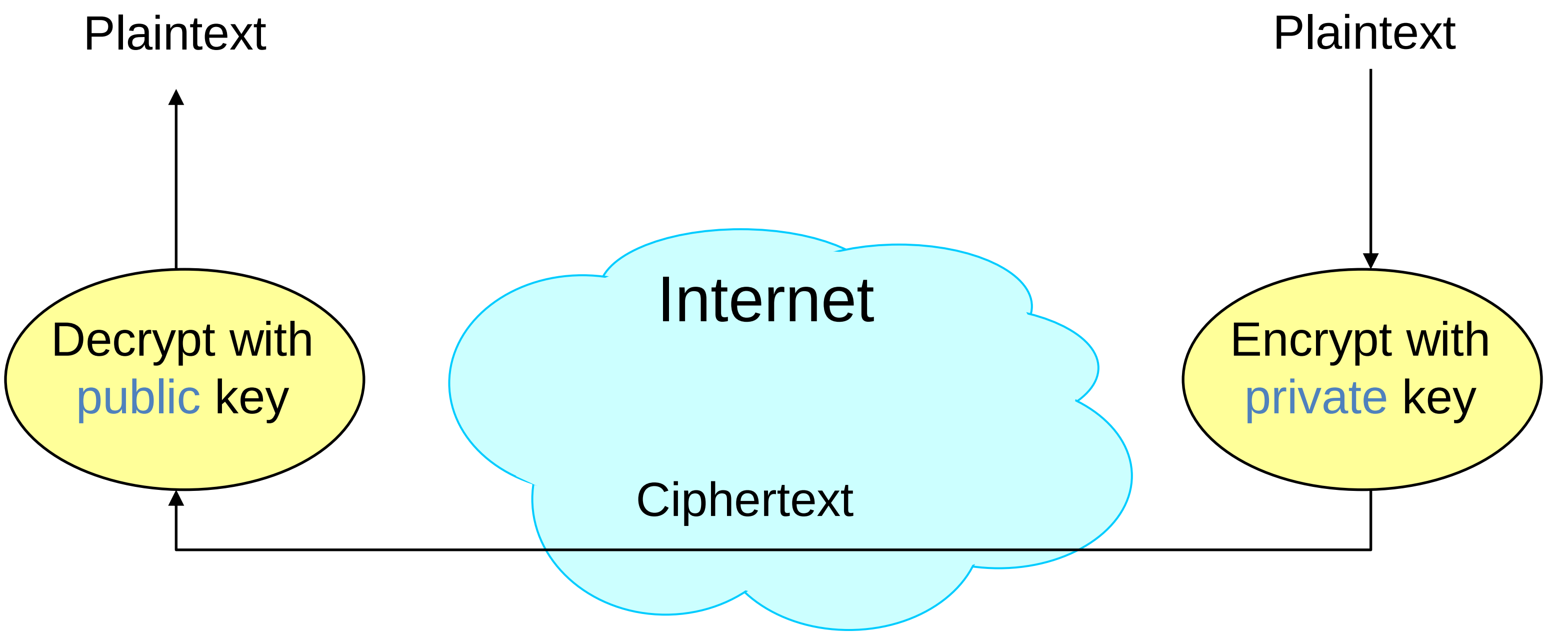
Public Key / Asymmetric Encryption

- **Sender uses receiver's public key**
  - Advertised to everyone
- **Receiver uses complementary private key**
  - Must be kept secret



Works in Reverse Direction Too!

- **Sender uses his own private key**
- **Receiver uses complementary public key**
- **Allows sender to prove he knows private key**



Realizing Public Key Cryptography

- **Invented in the 1970s**
  - *Revolutionized* cryptography
  - (Was actually invented earlier by British intelligence)
- **How can we construct an encryption/decryption algorithm with public/private properties?**
  - Answer: Number Theory
- **Most fully developed approach: RSA**
  - Rivest / Shamir / Adleman, 1977; RFC 3447
  - Based on modular multiplication of very large integers
  - Very widely used (e.g., SSL/TLS for **`https`**)

Cryptographic Toolkit

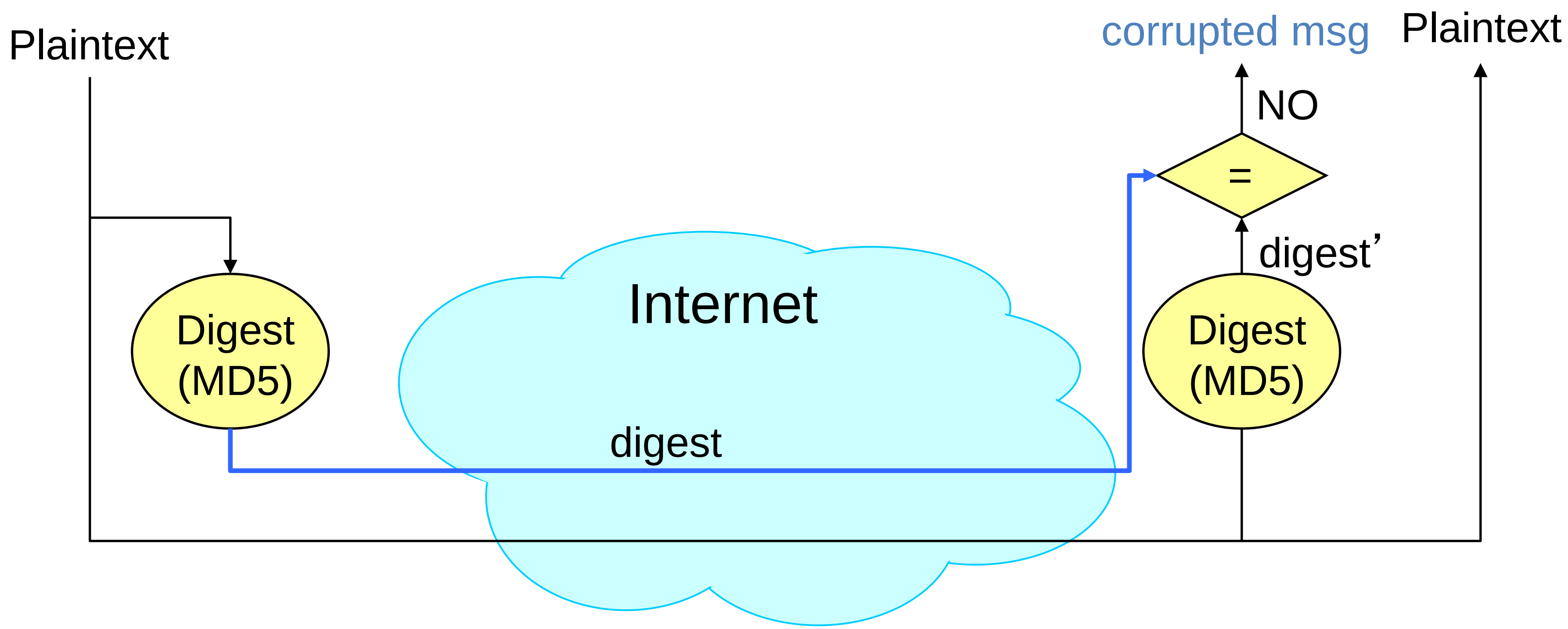
Cryptographic Toolkit

- **Confidentiality: Encryption**
- **Integrity: ?**
- **Authentication: ?**
- **Provenance: ?**

Integrity: Cryptographic Hashes

- **Sender computes a *digest* of message *m*, i.e., *H(m)***
  - H() is a publicly known *hash function (e.g., MD5, SHA-1)*
- **Send *m* in any manner**
- **Send digest *d = H(m)* to receiver in a secure way:**
  - Using another physical channel
  - Using encryption *(why does this help?)*
- **Upon receiving *m* and *d*, receiver re-computes *H(m)* to see whether result agrees with *d***

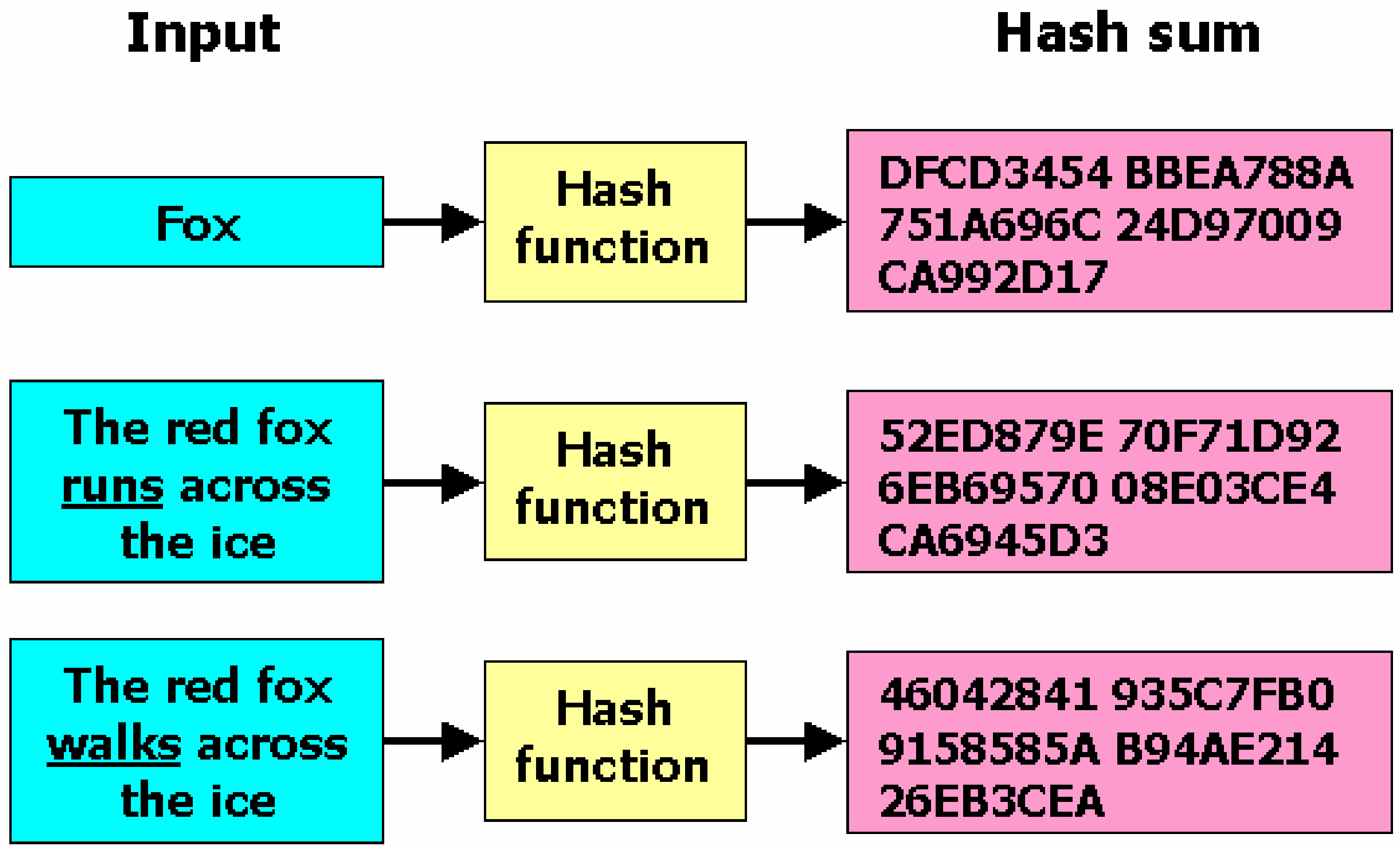
Operation of Hashing for Integrity



Cryptographically Strong Hashes

- **Hard to find collisions**
  - Adversary can’t find two inputs that produce same hash
  - Someone cannot alter message without modifying digest
  - Can succinctly refer to large objects
- **Hard to invert**
  - Given hash, adversary can’t find input that produces it
  - Can refer obliquely to private objects (e.g., passwords)
    - Send hash of object rather than object itself

Effects of Cryptographic Hashing

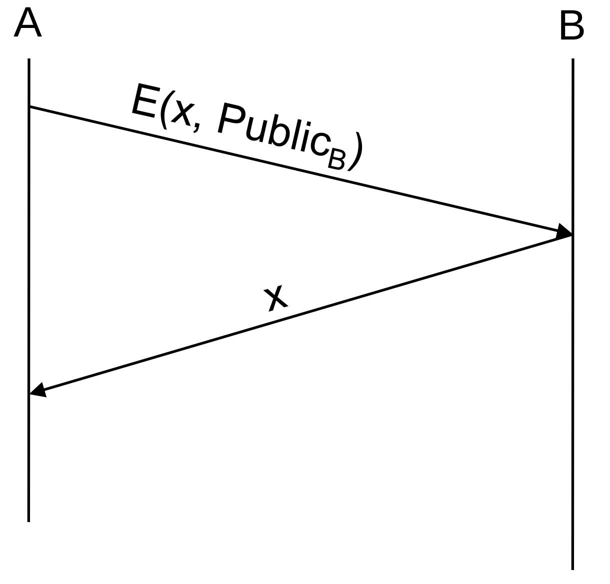


Cryptographic Toolkit

- **Confidentiality:** **Encryption**
- **Integrity:** **Cryptographic Hash**
- **Authentication:** **?**
- **Provenance:** **?**

Public Key Authentication

- **Each side need only to know the other side's public key**
  - No secret key need be shared
- **A encrypts a nonce (random number) *x* using B's public key**
- **B proves it can recover *x***
- **A can authenticate itself to B in the same way**



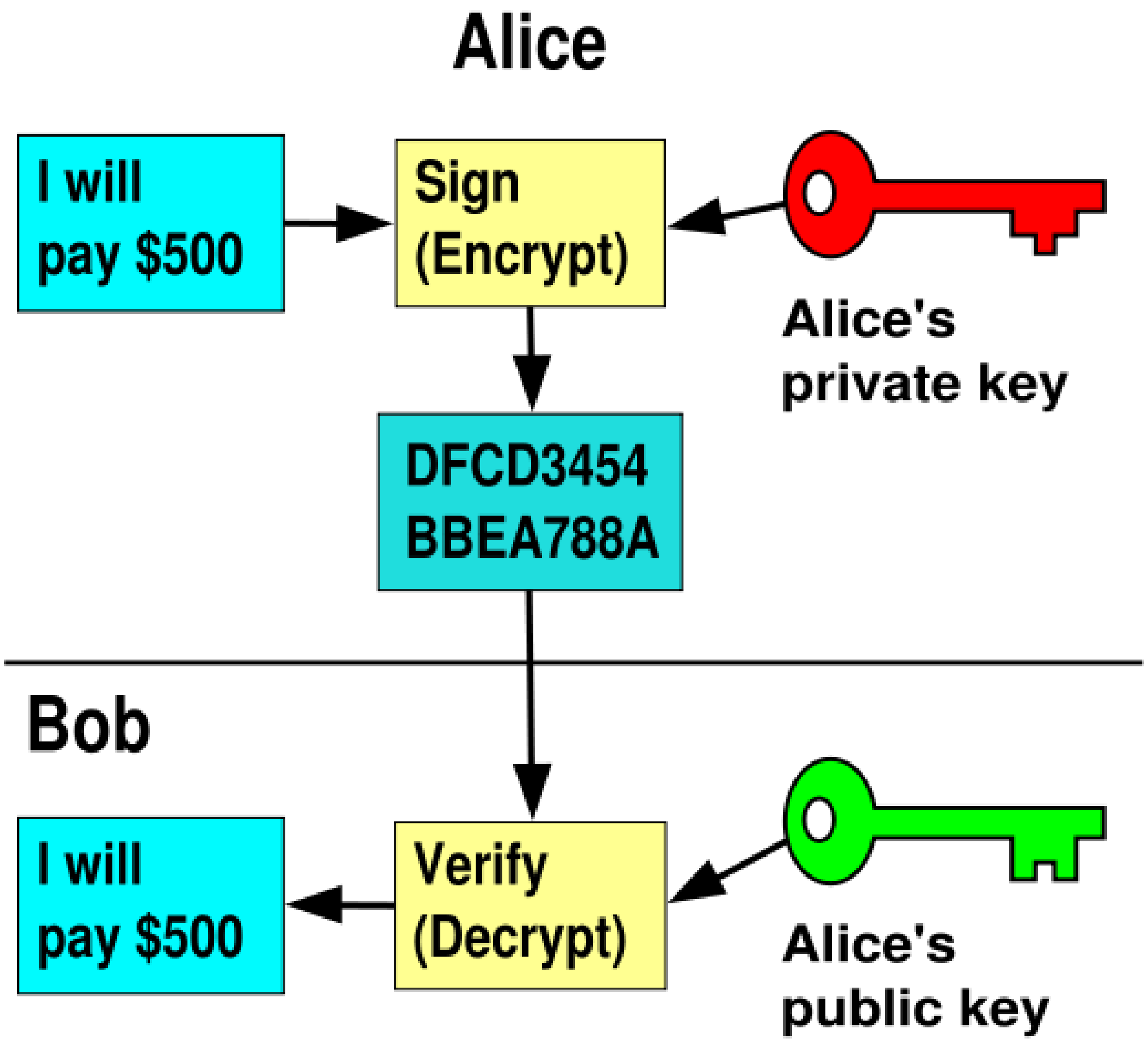
Cryptographic Toolkit

- **Confidentiality:** **Encryption**
- **Integrity:** **Cryptographic Hash**
- **Authentication:** **Decrypting nonce**
- **Provenance:** **?**

Digital Signatures

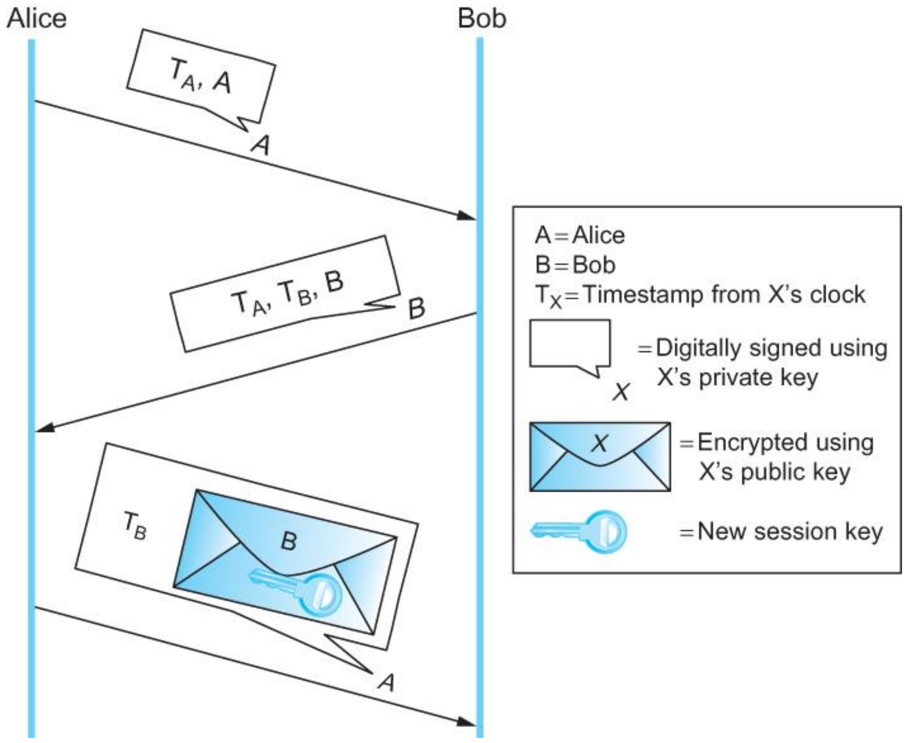
- **Suppose Alice has published public key K_E**
- **If she wishes to prove who she is, she can send a message *x* encrypted with her private key K_D**
  - Therefore: anyone w/ public key K_E can recover *x*, verify that Alice must have sent the message
  - It provides a digital signature
  - Alice can't deny later deny it $\Rightarrow$ non-repudiation

RSA Crypto & Signatures, con't



Key Pre Distribution

- **Pre-Distribution of Symmetric Keys**
  - Public Key Authentication Protocols



A public-key authentication protocol that does not depend on synchronization. Alice checks her own timestamp against her own clock, and likewise for Bob.

Summary of Our Crypto Toolkit

- **If we can securely distribute a key, then**
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- **Public key cryptography does away with problem of secure key distribution**
  - But not as computationally efficient
  - Often addressed by using public key crypto to exchange a session key
  - And not guaranteed secure
    - but **major** result if not

Summary of Our Crypto Toolkit, con't

- **Cryptographically strong hash functions provide major building block for integrity (e.g., SHA-1)**
  - As well as providing concise digests
  - And providing a way to prove you know something (e.g., passwords) without revealing it (non-invertibility)
  - But: worrisome recent results regarding their strength
- **Public key also gives us signatures**
  - Including sender <u>non-repudiation</u>
- **Turns out there's a crypto trick based on similar algorithms that allows two parties *who don't know each other's public key* to securely negotiate a secret key even in the presence of eavesdroppers**

PKIs and HTTPS

Public Key Infrastructure *(PKI)*

- **Public key crypto is *very* powerful …**
- **… but the realities of tying public keys to real world identities turn out to be quite hard**

- **PKI: *Trust distribution* mechanism**
  - Authentication via Digital Certificates
- **Trust doesn't mean someone is honest, just that they are who they say they are…**

Managing Trust

- **The most solid level of trust is rooted in our direct personal experience**
  - E.g., Alice's trust that Bob is who they say they are
  - Clearly doesn't scale to a global network!

- **In its absence, we rely on *delegation***
  - Alice trusts Bob's identity because Charlie attests to it ….
  - …. and Alice trusts Charlie

Managing Trust, con't

- **Trust is not particularly transitive**
  - Should Alice trust Bob because she trusts Charlie …
  - … and Charlie vouches for Donna …
  - … and Donna says Eve is trustworthy …
  - … and Eve vouches for Bob's identity?

- **Two models of delegating trust**
  - Rely on your set of friends and their friends
    - "Web of trust" -- e.g., PGP
  - Rely on trusted, well-known authorities (*and their minions*)
    - "Trusted root" -- e.g., HTTPS

PKI Conceptual Framework

- **Trusted-Root PKI:**
  - Basis: well-known public key serves as root of a hierarchy
  - Managed by a Certificate Authority (CA)
- **To publish a public key, ask the CA to digitally sign a statement indicating that they agree (“certify”) that it is indeed your key**
  - This is a certificate for your key (*certificate* = bunch of bits)
    - Includes both your public key and the signed statement
  - Anyone can verify the signature
- **Delegation of trust to the CA**
  - They’d better not screw up (duped into signing bogus key)
  - They’d better have procedures for dealing with stolen keys
  - Note: can build up a hierarchy of signing

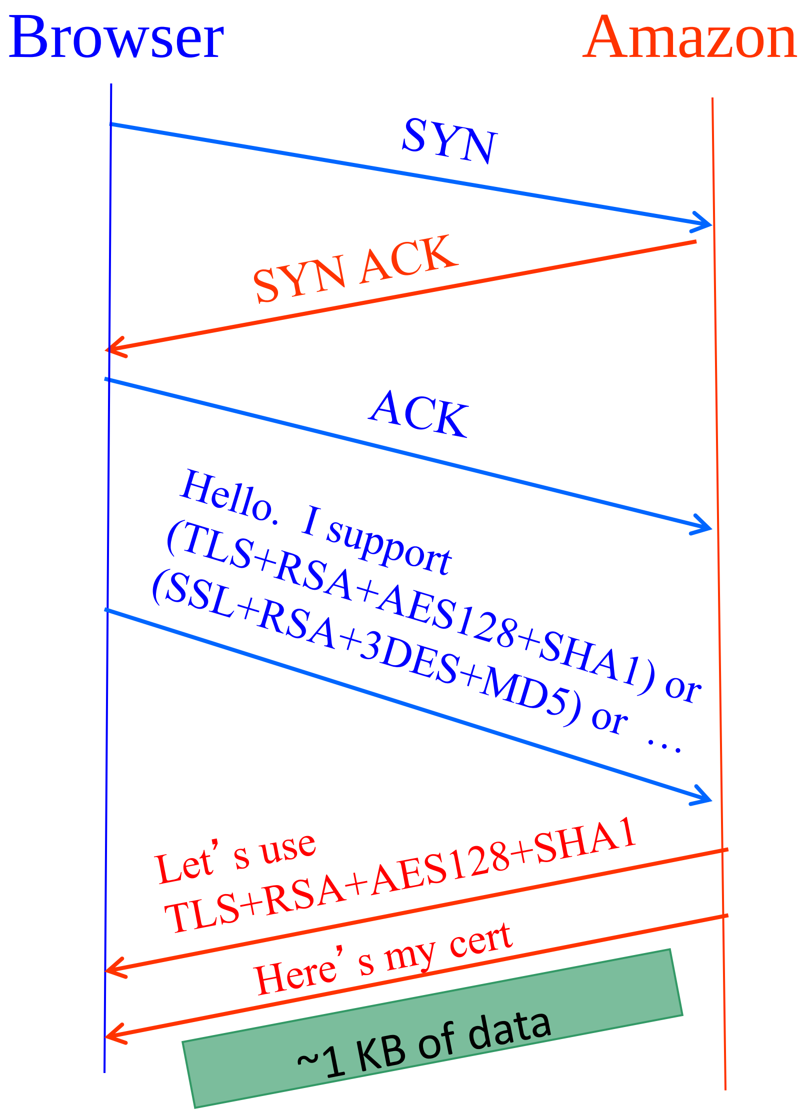
Putting It All Together: HTTPS

- **Steps after clicking on https://www.amazon.com**
- **https = "Use HTTP over SSL/TLS"**
  - SSL = Secure Socket Layer
  - TLS = Transport Layer Security
    - Successor to SSL, and compatible with it
  - RFC 4346
- **Provides security layer (authentication, encryption) on top of TCP**
  - Fairly transparent to the app

Application (e.g., HTTP)
Secure Transport Layer
TCP
IP
Link

HTTPS Connection (SSL/TLS), con’t

- **Browser (client) connects via TCP to Amazon’s `HTTPS` server**
- **Client sends over list of crypto protocols it supports**
- **Server picks protocols to use for this session**
- **Server sends over its certificate**
- **(all of this is in the clear)**



Inside the Server's Certificate

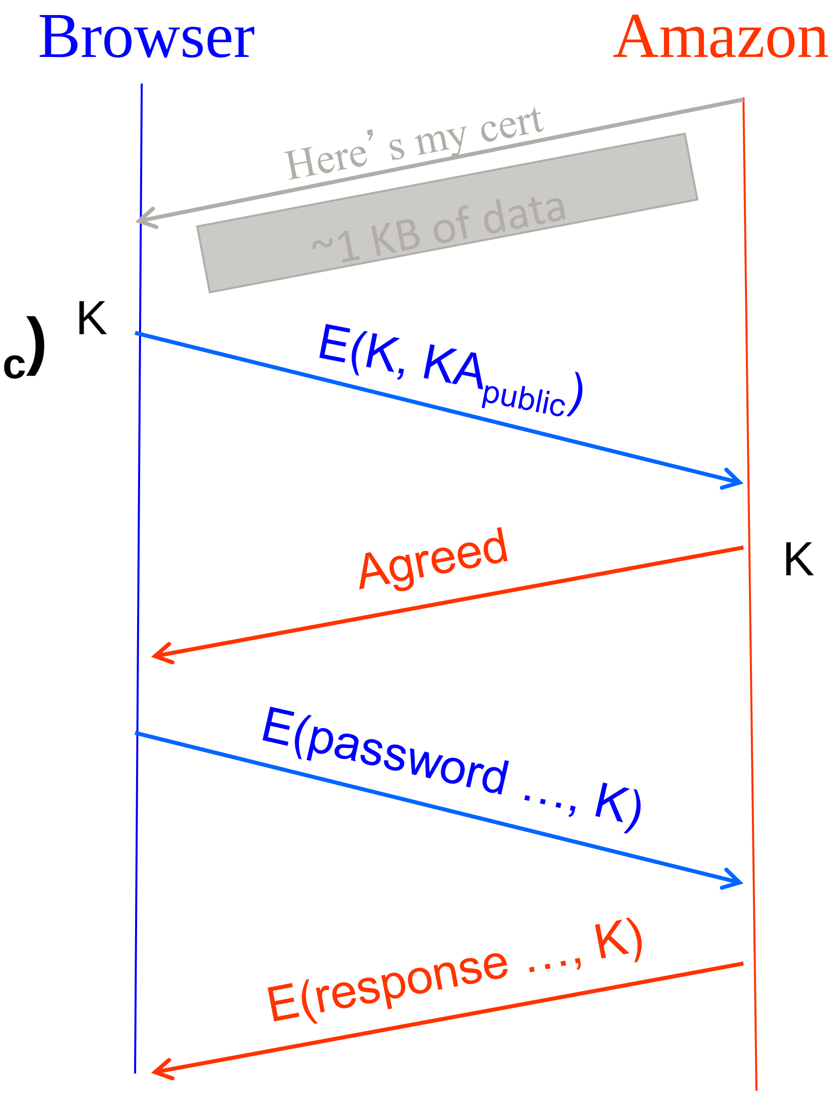
- **Name associated with cert (e.g., Amazon)**
- **Amazon's public key**
- **A bunch of auxiliary info (physical address, type of cert, expiration time)**
- **URL to *revocation center* to check for revoked keys**
- **Name of certificate's signatory (who signed it)**
- **A public-key signature of a hash (MD5) of all this**
  - Constructed using the signatory's private RSA key

Validating Amazon's Identity

- **Browser retrieves cert belonging to the signatory**
  - These are hardwired into the browser
- **If it can't find the cert, then warns the user that site has not been verified**
  - And may ask whether to continue
  - Note, can still proceed, just without authentication
- **Browser uses public key in signatory's cert to decrypt signature**
  - Compares with its own MD5 hash of Amazon's cert
- **Assuming signature matches, now have high confidence it's indeed Amazon …**
  - … assuming signatory is trustworthy

HTTPS Connection (SSL/TLS), con’t

- **Browser constructs a random *session key* K**
- **Browser encrypts K using Amazon’s public key**
- **Browser sends E(K, KA_{public}) to server**
- **Browser displays** 
- **All subsequent communication encrypted w/ symmetric cipher using key K**
  - E.g., client can authenticate using a password



Next Class

- **Some new trends, Software-Defined Networking**
- **Second-to-last class!**