Introduction to Computer Networks

COSC 4377

Lecture 23

Spring 2012
April 16, 2012

Announcements

- HW11 due this week
- Exam 2 next week

HW11

- DNS Server
- Learn how to read an RFC
- Learn how to implement a server based on RFC

Today's Topics

- Security
  - Encryption
  - Integrity
  - Authentication
  - Certificate
  - HTTPS
  - Pharming

Confidentiality through Cryptography

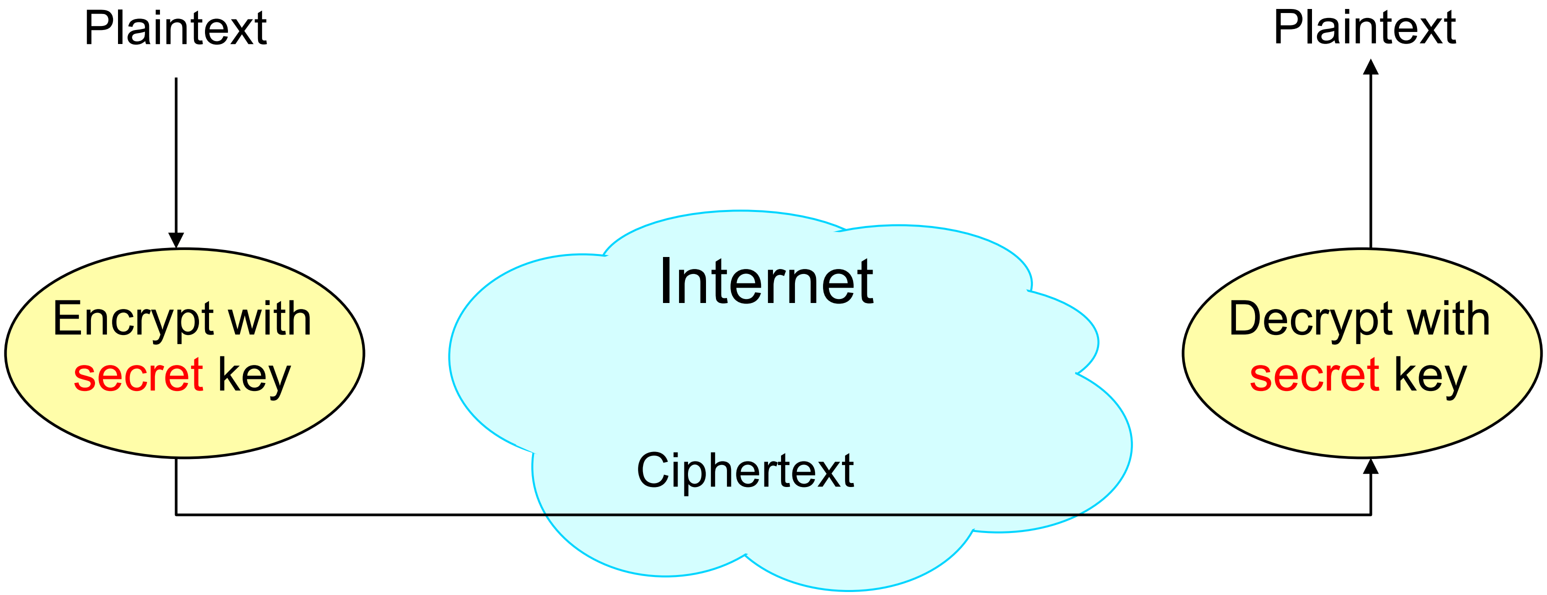
- Cryptography: *communication over insecure channel in the presence of adversaries*
- Central goal: how to encode information so that an adversary can't extract it …but a friend can
- General premise: a *key* is required for decoding
  - Give it to friends, keep it away from attackers
- Two different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

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

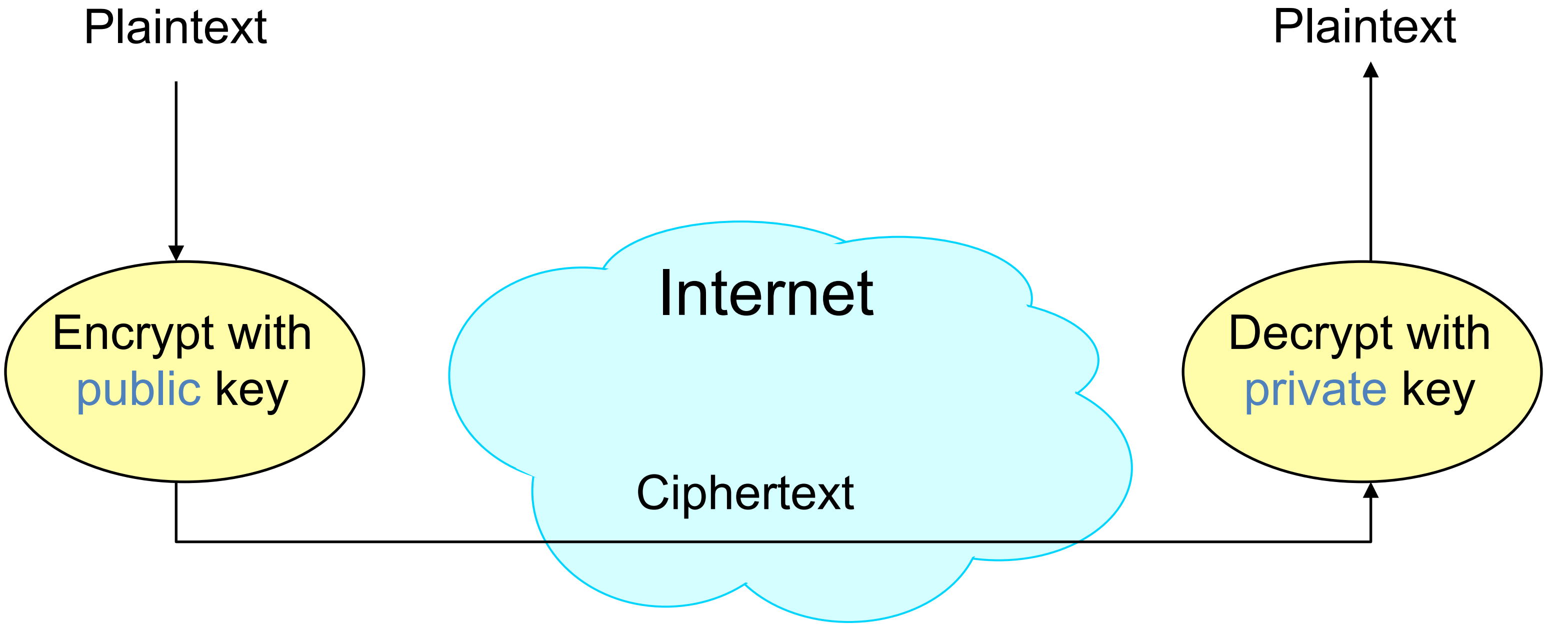


Asymmetric Encryption *(Public Key)*

- 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)

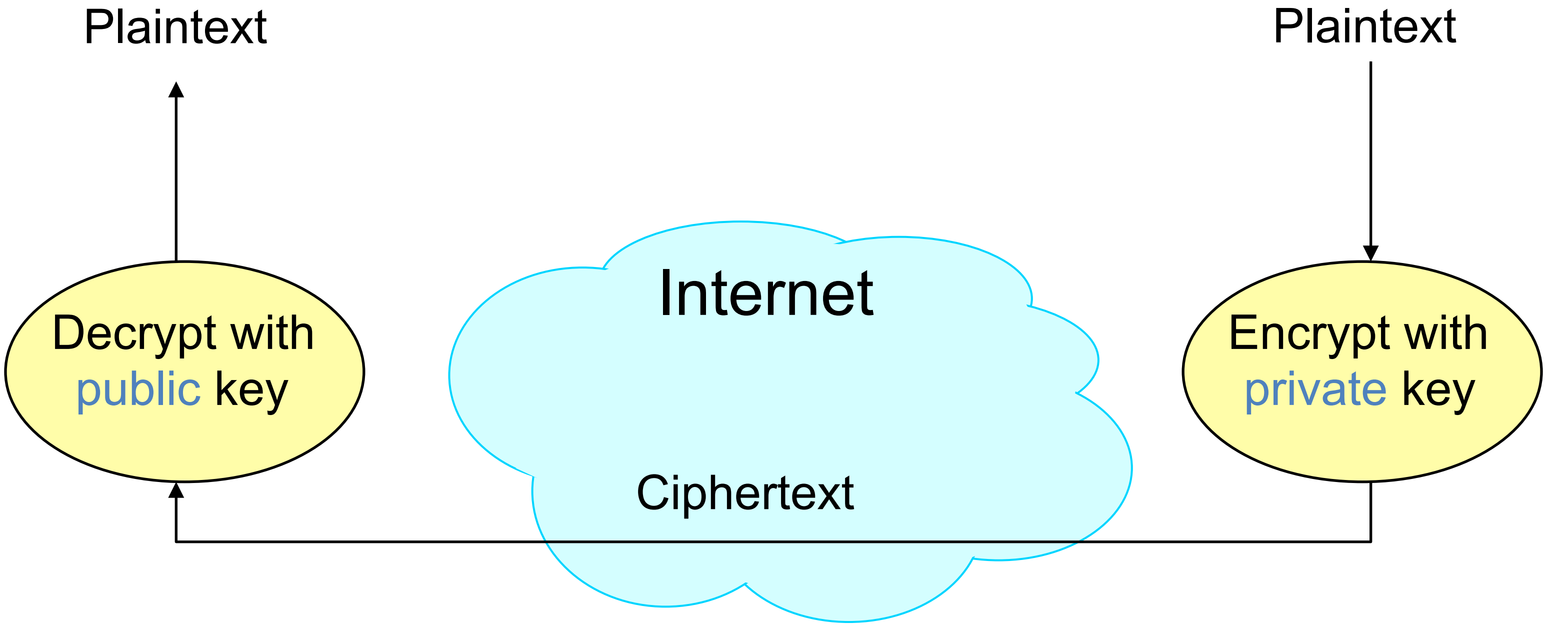
Public Key / Asymmetric Encryption

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



Works in Reverse Direction Too!

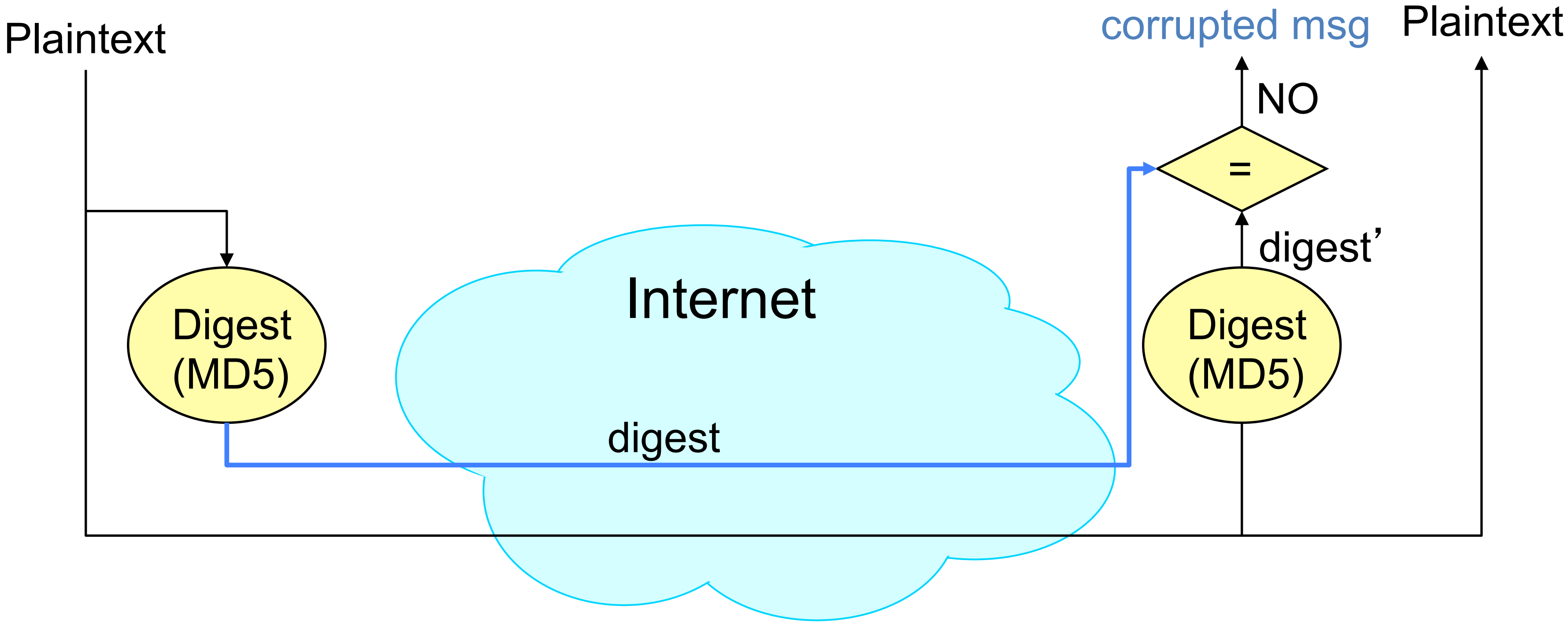
- Sender uses his own private key
- Receiver uses complementary public key
- Allows sender to prove he knows private key



Integrity: Cryptographic Hashes

- Sender computes a *digest* of message ***m***, i.e., *H(m)*
  - H() is a publicly known *hash function*
- 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***

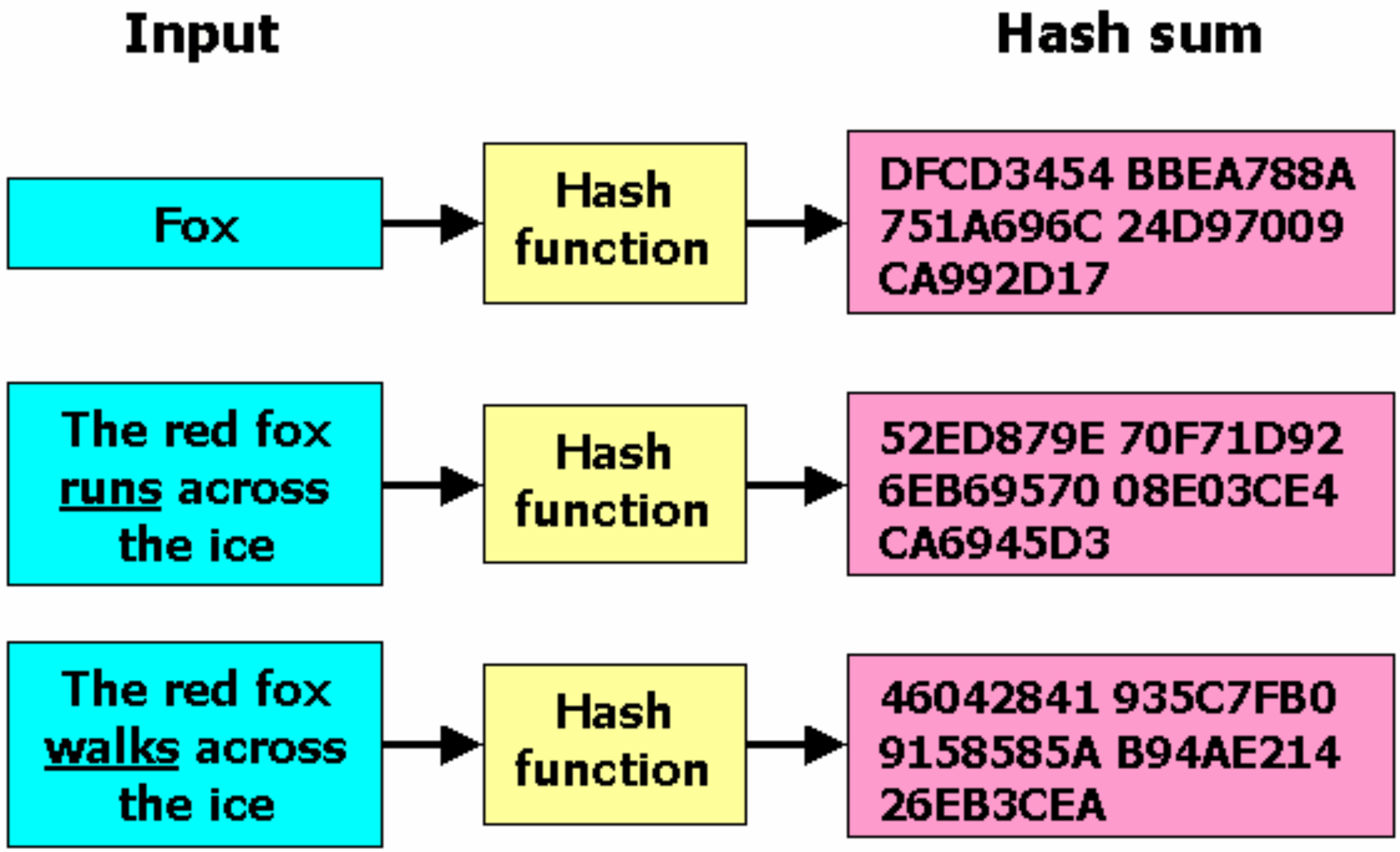
Operation of Hashing for Integrity



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



Input

Hash sum

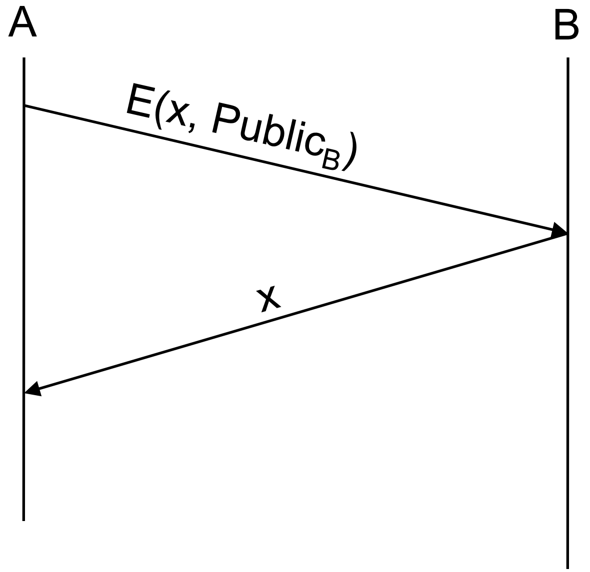
Fox → Hash function → DFCD3454 BBEA788A 751A696C 24D97009 CA992D17

The red fox runs across the ice → Hash function → 52ED879E 70F71D92 6EB69570 08E03CE4 CA6945D3

The red fox walks across the ice → Hash function → 46042841 935C7FB0 9158585A B94AE214 26EB3CEA

Public Key Authentication

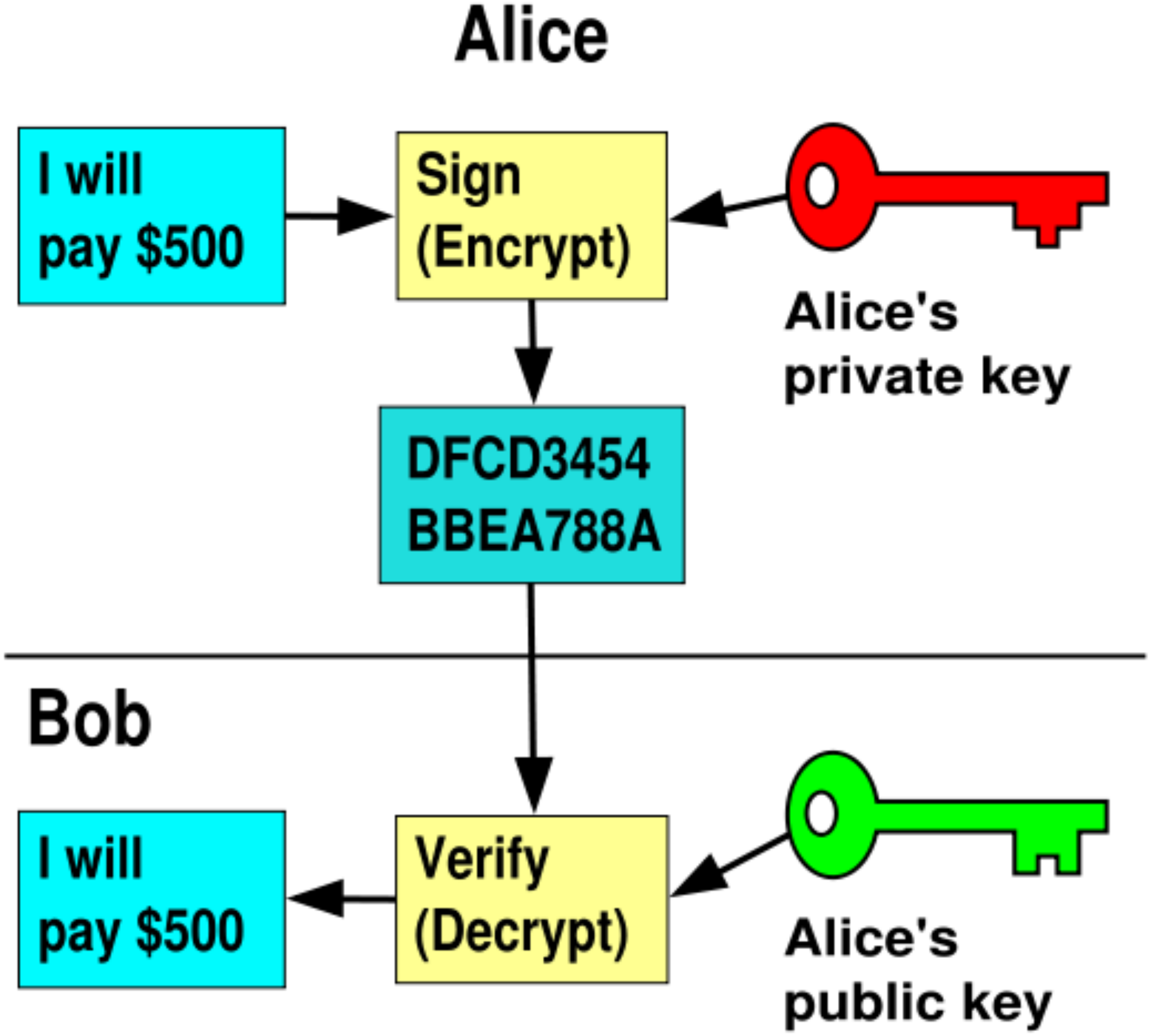
- Each side need only to know the other side's public key
  - No secret key need be shared
- **A** encrypts a nonce (random number) ***x*** using B's public key
- **B** proves it can recover ***x***
- **A** can authenticate itself to **B** in the same way



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 ⇒ non-repudiation

RSA Crypto & Signatures, con't



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

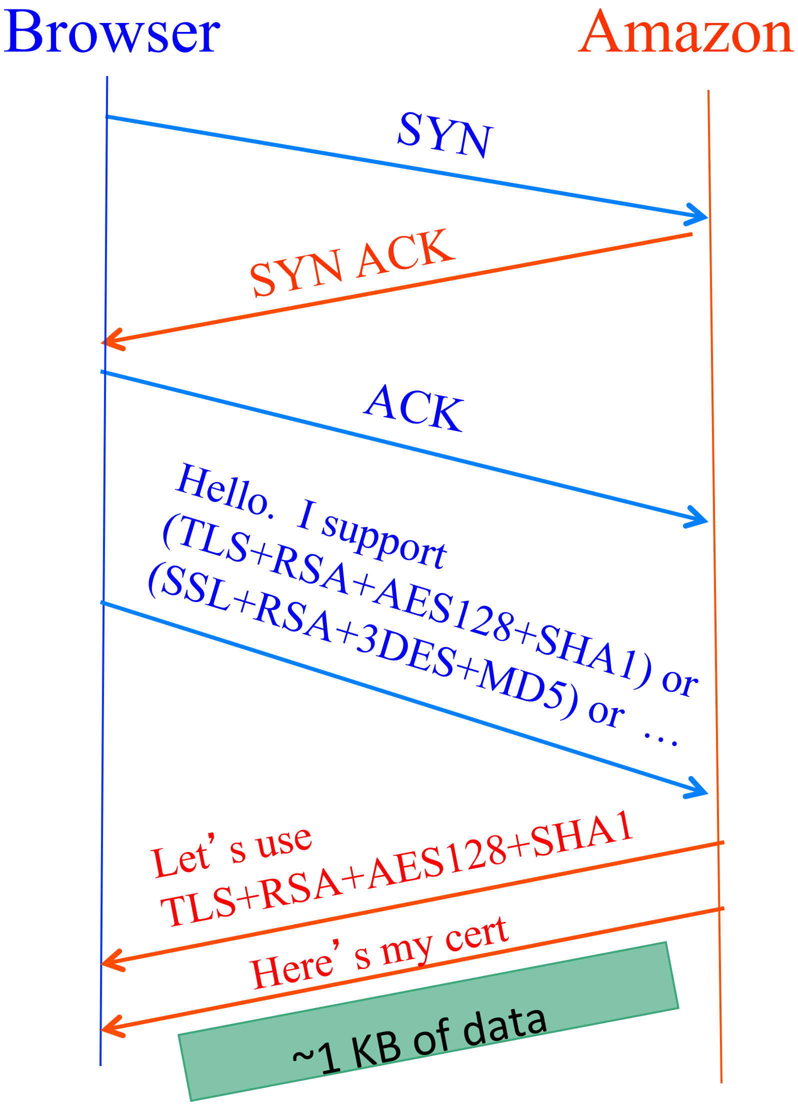
- Trusted-Root PKI:
  - Basis: well-known public key serves as root of a hierarchy
  - Managed by a Certificate Authority (CA)
- To publish a public key, ask the CA to digitally sign a statement indicating that they agree (“certify”) that it is indeed your key
  - This is a certificate for your key (*certificate* = 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

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

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

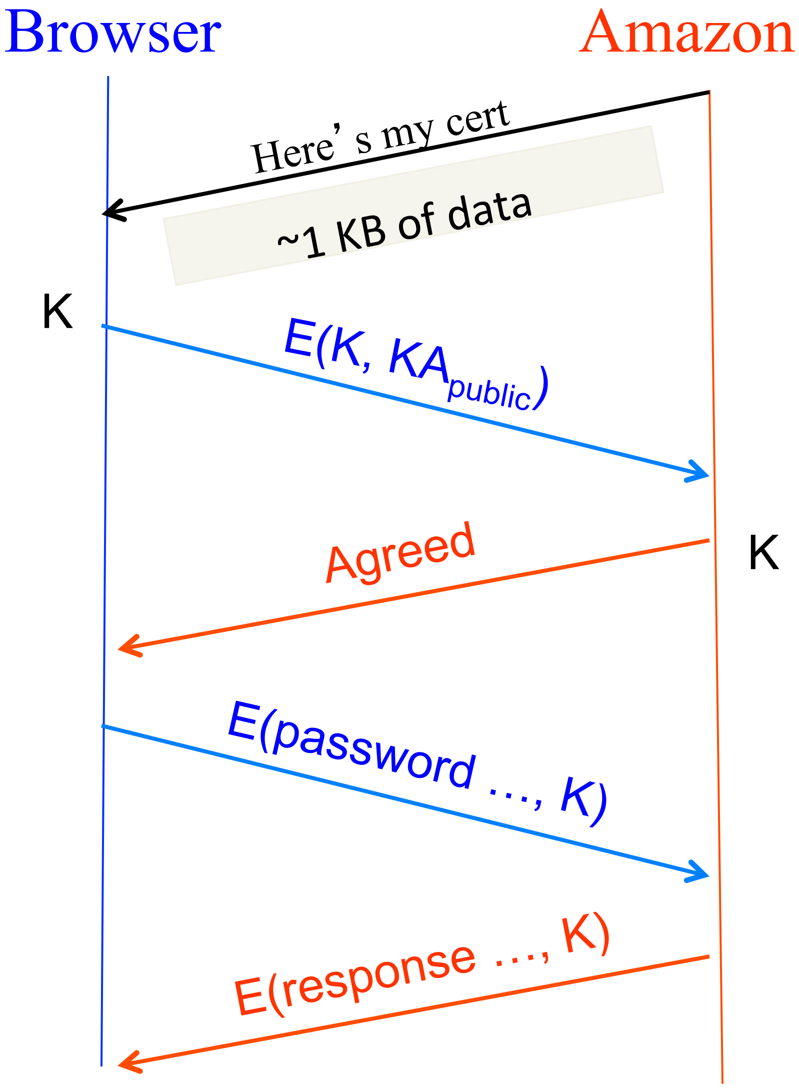
- **Name** associated with cert (e.g., Amazon)
- Amazon's **public key**
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- URL to *revocation center* to check for revoked keys
- Name of certificate's **signatory** (who signed it)
- A public-key **signature** of a hash (MD5)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

Validating Amazon's Identity

- Example: certificate of entity Amazon
  Cert = E({Amazon, KAmazon$_{public}$}, KCA$_{private}$)
- 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 …
  - … <u>assuming signatory is trustworthy</u>

`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



Pharming

- How can we get web clients to redirect to malicious sites?
- Name resolution
  - Send a query to a DNS
  - Trust the IP address returned by the DNS
  - Other ways to go from name to IP?