Network security

Security is a problem

- **Networks are**
  - shared by many with differing goals and interests
- **No security = potential compromise!**
  - Exposure of your information
  - Encryption is not enough!
  - Still need data integrity, originality, and timeliness!

Basic Requirements for Secure Communication

- **Availability**
  - Will the network deliver data?
  - Infrastructure compromise, DDoS
- **Authentication**
  - Who is this person/machine?
  - Spoofing, phishing
- **Integrity**
  - Do messages arrive in original form?

Basic Requirements for Secure Communication

- **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 user/machine 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

Other Desirable Security Properties

- **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?

Security in the internet

- Focus on basic requirements
- Simple cryptographic methods
- Cryptographic toolkit (Hash, Digital Signature, …)
- PKIs and HTTPS
- Compromises, worms, and underground market
- Dealing with DDoS

Basic Forms of Cryptography

Confidentiality through Cryptography

- **Cryptography**
  - Communication over insecure channel in the presence of adversaries

- **Studied for thousands of years**
  - See Singh's The Code Book for an excellent history

- **Central goal**
  - How to encode information so that an adversary can't extract it …but a friend can

Confidentiality through Cryptography

- **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

Plaintext → Encrypt with **secret** key → Ciphertext → Internet → Decrypt with **secret** key → Plaintext
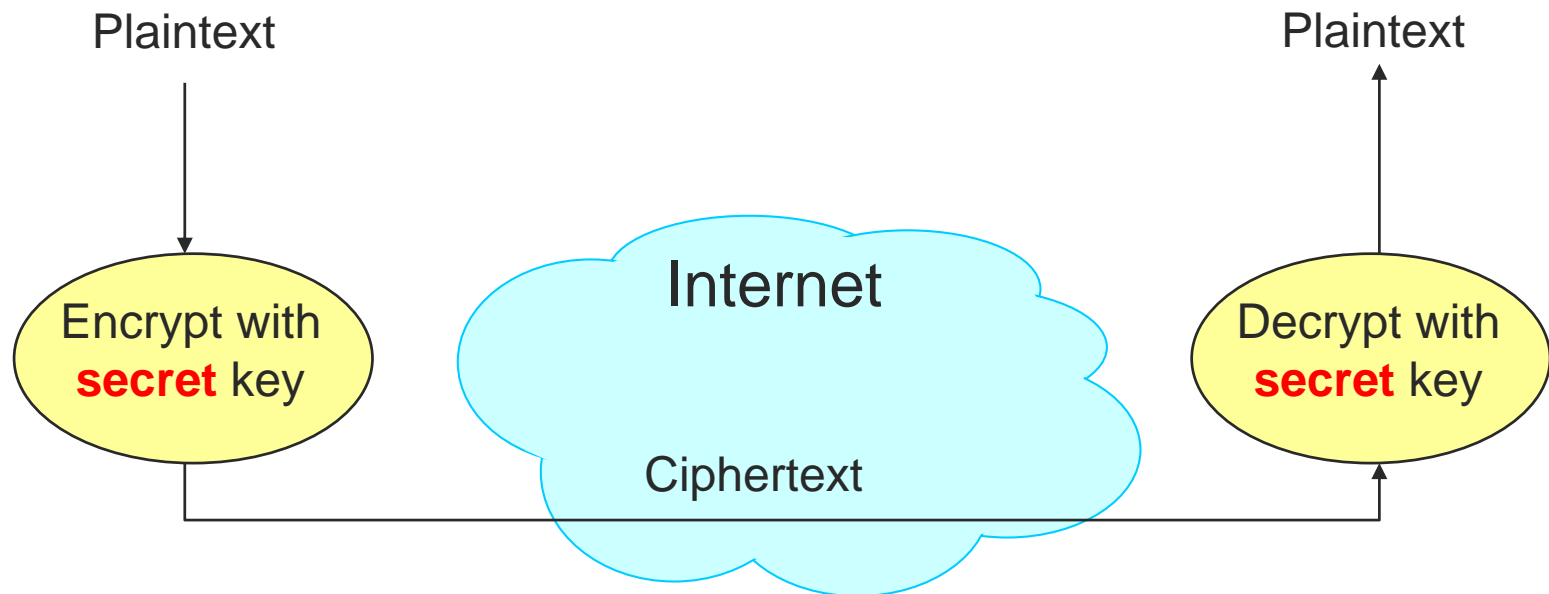
Asymmetric Encryption (Public Key)

- **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 e!

Asymmetric Encryption (Public Key)

- **Alice** wants to send to **Bob**
  - Fetch Bob's public key (say from Bob's home page)
  - Encrypt message with Bob's public key
  - 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

Plaintext

Plaintext

Encrypt with **public** key

Internet

Ciphertext

Decrypt with **private** key
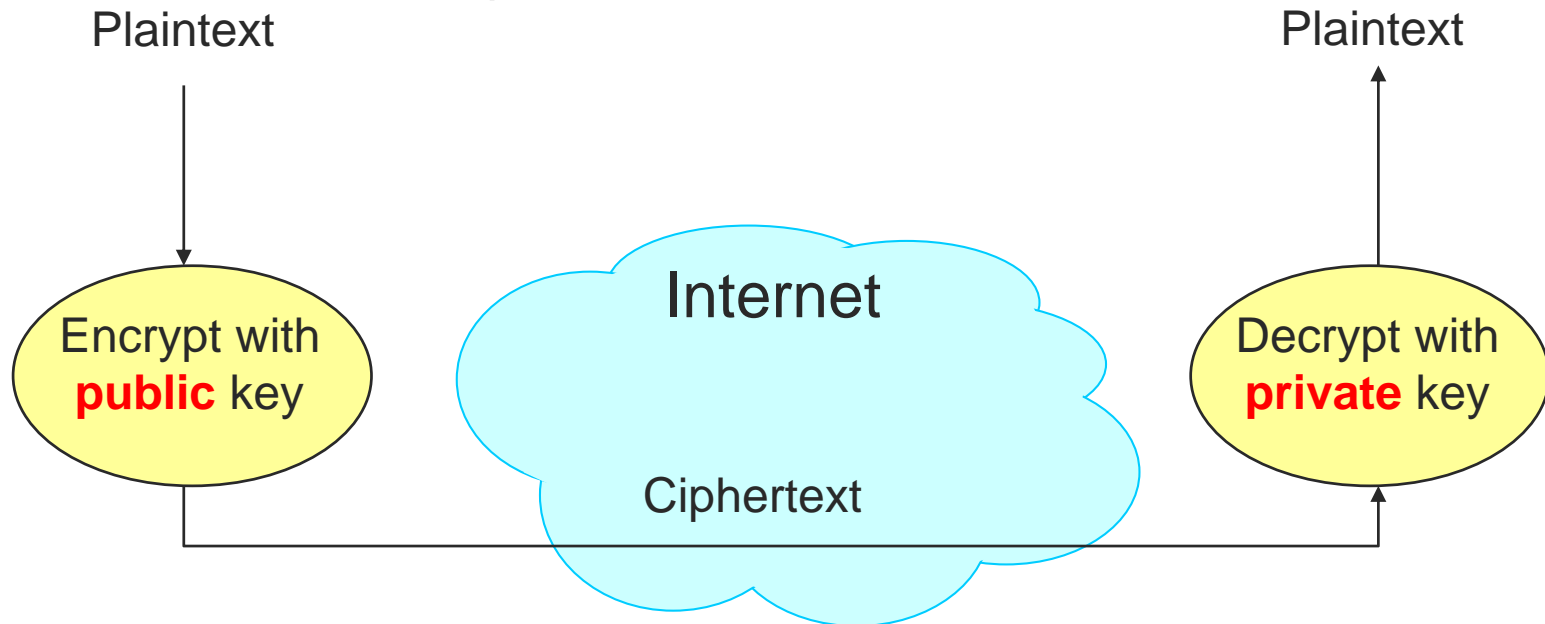
Works in Reverse Direction Too!

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

Plaintext

Plaintext

Decrypt with **public** key

Internet

Ciphertext

Encrypt with **private** key
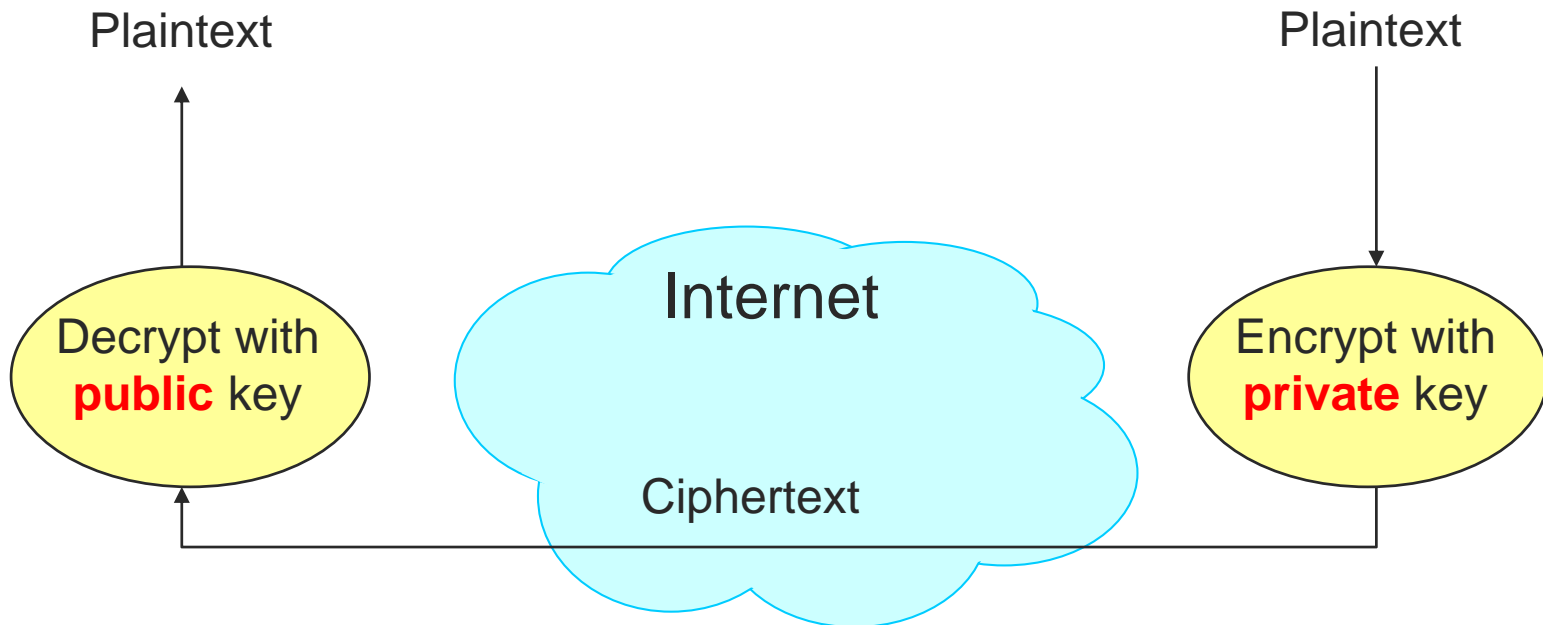
Realizing Public Key Cryptography

- Invented in the 1970s (probably even as early as the 1960s)
  - Revolutionized cryptography
- 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

- 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
- **Send *m* in any manner**
- **Send digest *d* = *H(m)* to receiver in a secure way**
  - Using another physical channel
  - Using encryption
- **Receive *m* and *d***
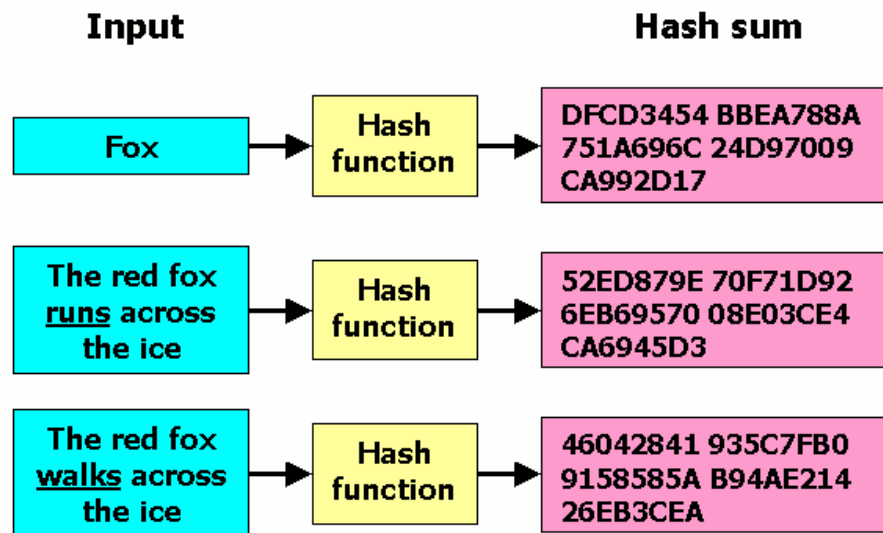 - Receiver re-computes *H(m)* to see whether result agrees with *d*

Operation of Hashing for Integrity

Plaintext

corrupted msg   Plaintext

NO

=

digest'

Internet

Digest
(MD5)
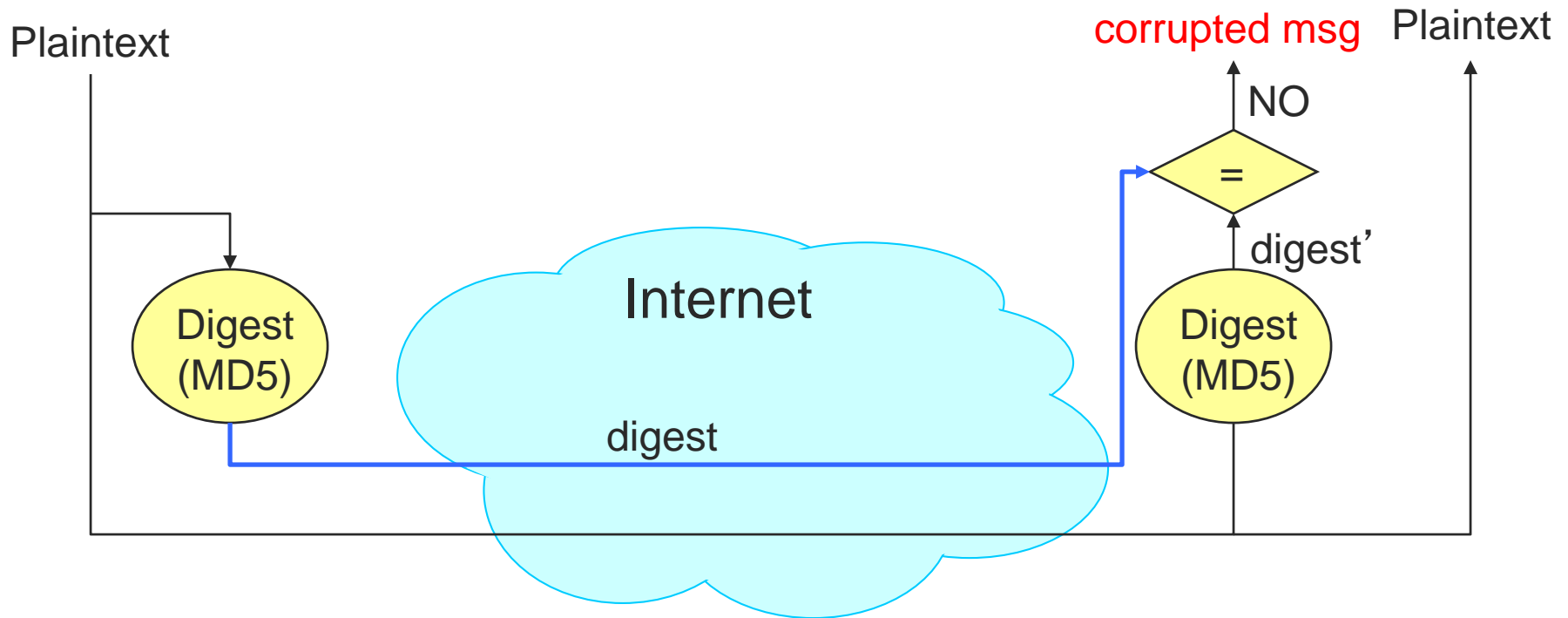
Digest
(MD5)

digest

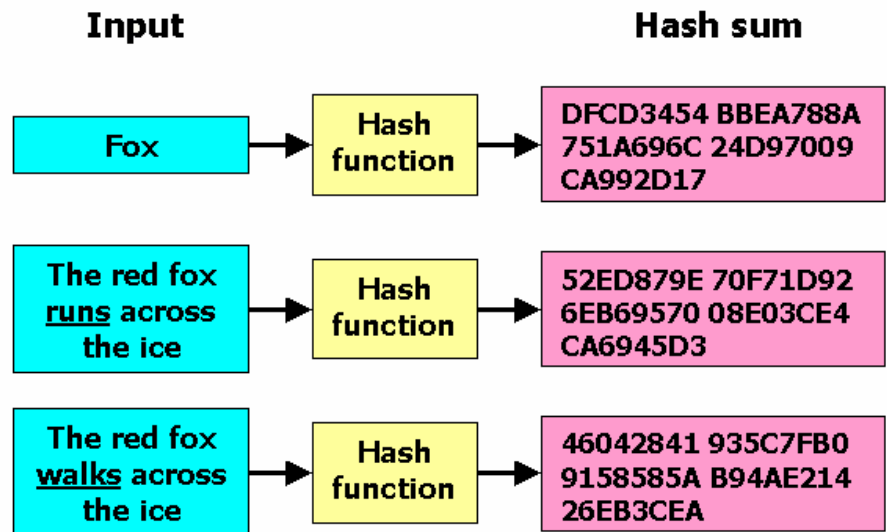# Cryptographically Strong Hashes

- Hard to find collisions
  - Adversary can't find two inputs that produce same hash
  - Hard to alter message without modifying digest
  - Can succinctly refer to large objects

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

Cryptographically Strong Hashes

- 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

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

Cryptographic Toolkit

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

Public Key Authentication

- Each party only knows the other'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
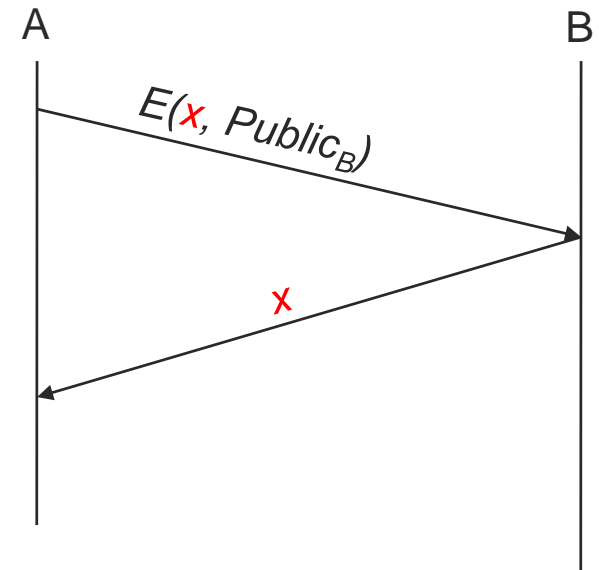
A $\qquad\qquad\qquad\qquad\qquad$ B

$E(x, Public_B)$

x

Cryptographic Toolkit

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

Digital Signatures

- Alice publishes public key *KE*
- Prove she is Alice!
  - Send a message *x* encrypted with her private key *KD*
  - Anyone w/ public key *KE* 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
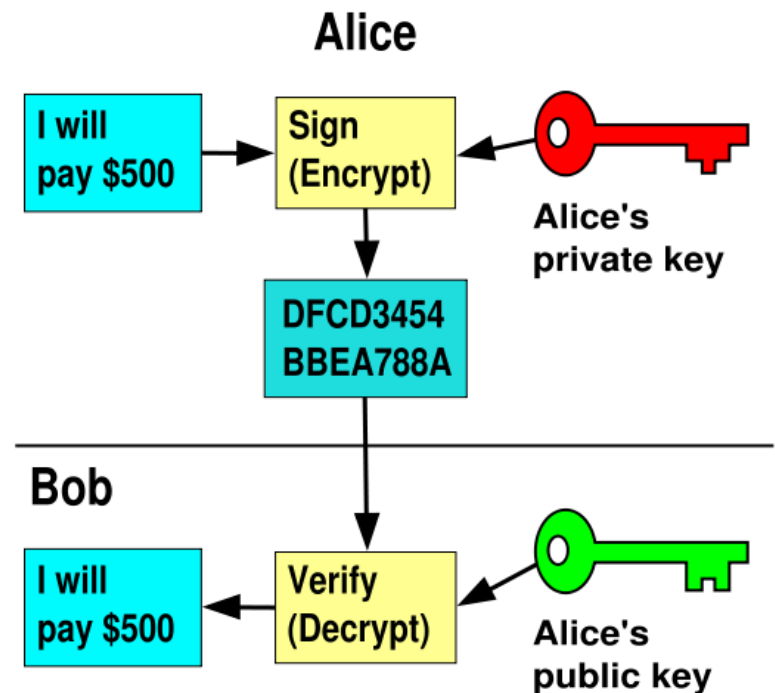
Our Crypto Toolkit

- **Secure key distribution**
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality

- **Public key cryptography**
  - No need of secure key distribution
  - But not as computationally efficient
    - Often addressed by using public key crypto to exchange a session key then used for symmetric crypto
  - Not guaranteed secure
    - but major result if not

Our Crypto Toolkit

- Cryptographically strong hash functions
  - Building block for integrity (e.g., SHA-1, SHA-2)
  - 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 non-repudiation

What is Missing?

- How can you relate a key to a person?
  - Trust (PKIs)

- How do all these pieces fit together?
  - SSL

- What about availability?

Public Key Infrastructure (PKI)

- Public key crypto is very powerful
  - But tying public keys to real world identities is 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 Bob
  - 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 completely 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)
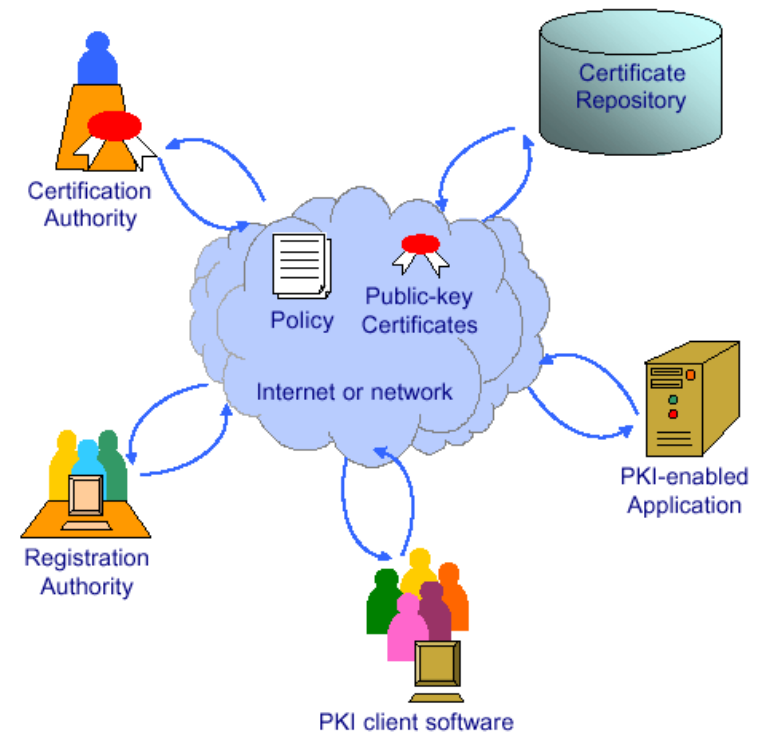
PKI Conceptual Framework

- Publishing a public key
  - CA digitally signs statement indicating that they agree ("certify") that it is indeed your key
  - This bunch of bits is a certificate for your key
    - Includes both your public key and the signed statement
  - Anyone that knows CA's public key can verify the signature
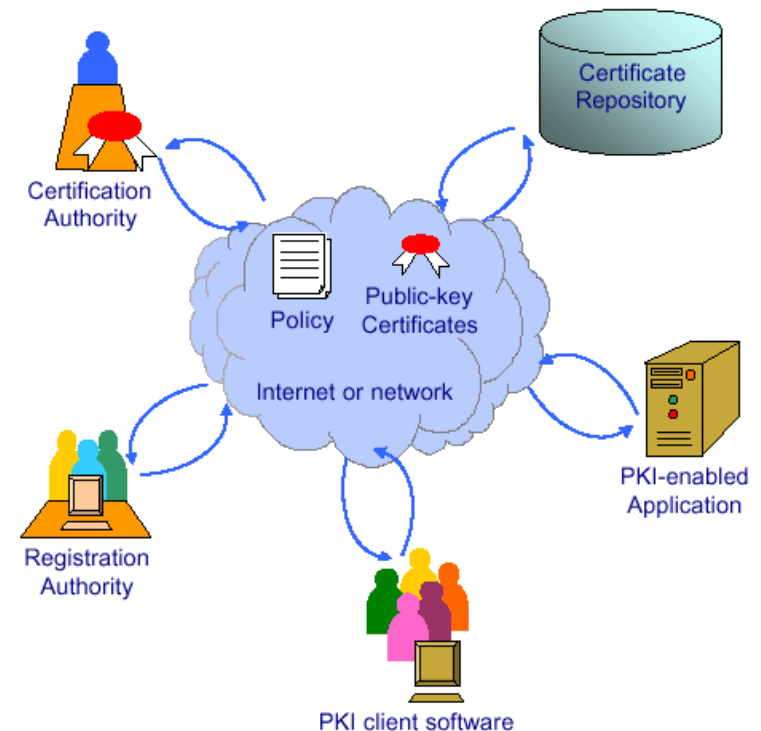
PKI Conceptual Framework

- 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
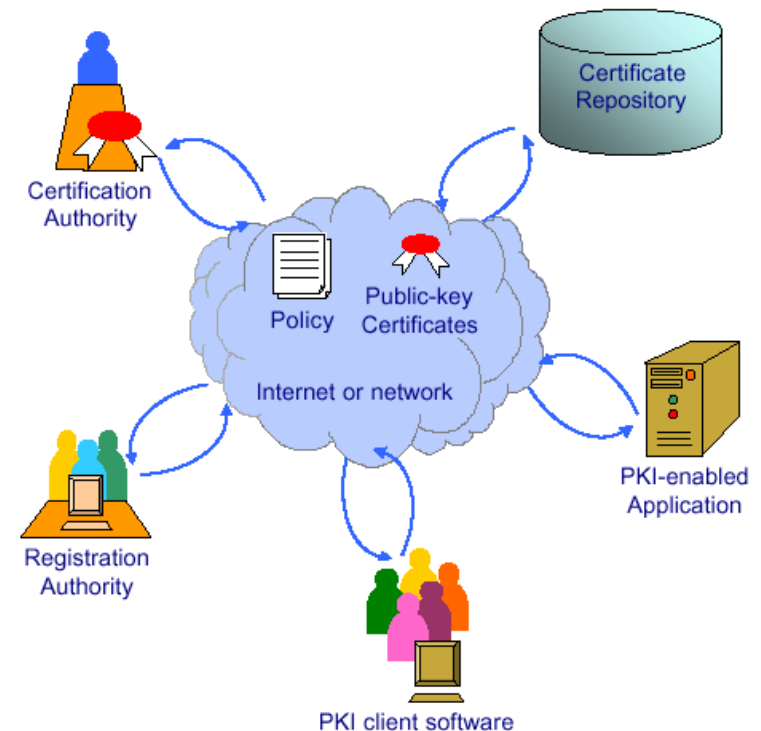
Digital Certificate

- Binds an entity with its corresponding public key
  - Signed by a recognized and trusted authority, i.e., Certification Authority (CA)
  - Provide assurance that a particular public key belongs to a specific entity

Digital Certificate

- Example: certificate of entity *Y*
  Cert = E({nameY, KYpublic}, KCAprivate)
  - *KCAprivate*: private key of Certificate Authority
  - *nameY*: name of entity *Y*
  - *KYpublic*: public key of entity *Y*
    - In fact, they may sign whatever bits you give them
- Your browser has a bunch of CAs

Certification Authority

- People, processes responsible for creation, delivery and management of digital certificates
- Organized in an hierarchy
  - To verify signature chain, follow hierarchy up to root
- Need to trust the CA's internal security
  - Not always a good idea … http://goo.gl/84l3i

CA-1   Root CA   CA-2
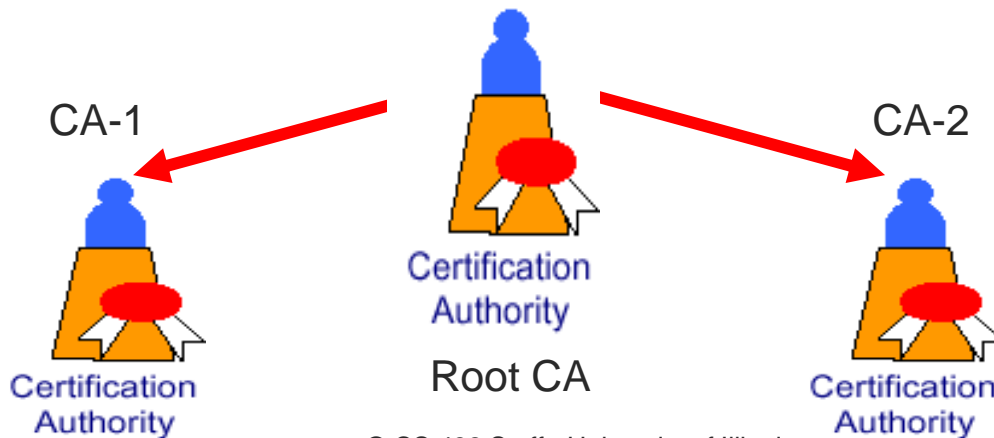
Registration Authority

- People & processes responsible for
  - Authenticating the identity of new entities (users or computing devices),
    - e.g. by phone, or physical presence + ID
  - Issuing requests to CA for certificates

- The CA must trust the Registration Authority

Certificate Repository

- A database accessible to all users of a PKI
- Contains
  - Digital certificates
  - Policy information associated with certs
  - Certificate revocation information
    - Vital to be able to identify certs that have been compromised
    - Usually done via a *revocation list*

Certificate Repository

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
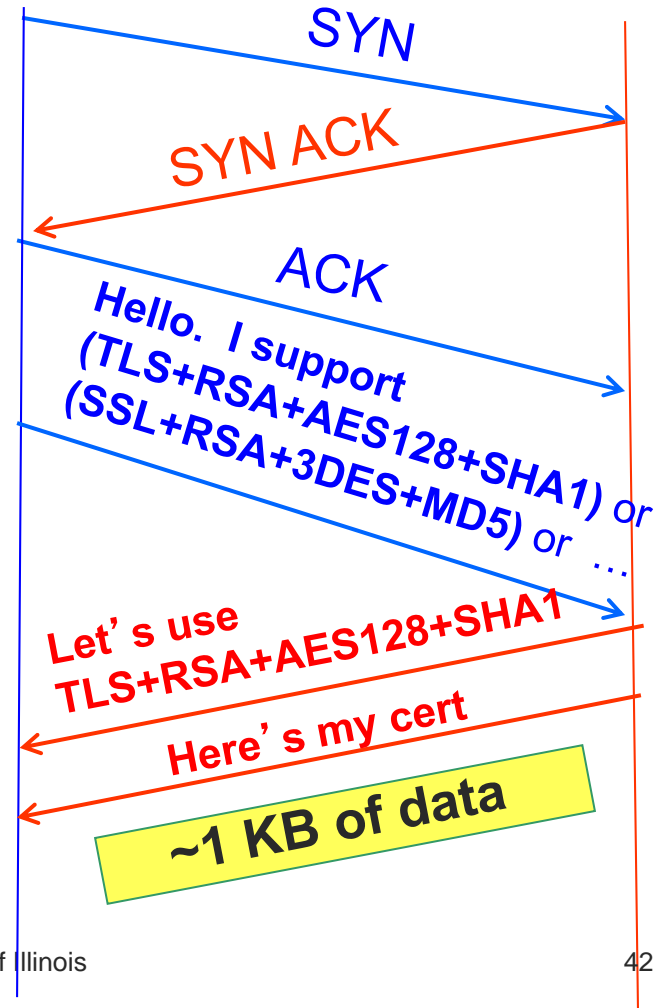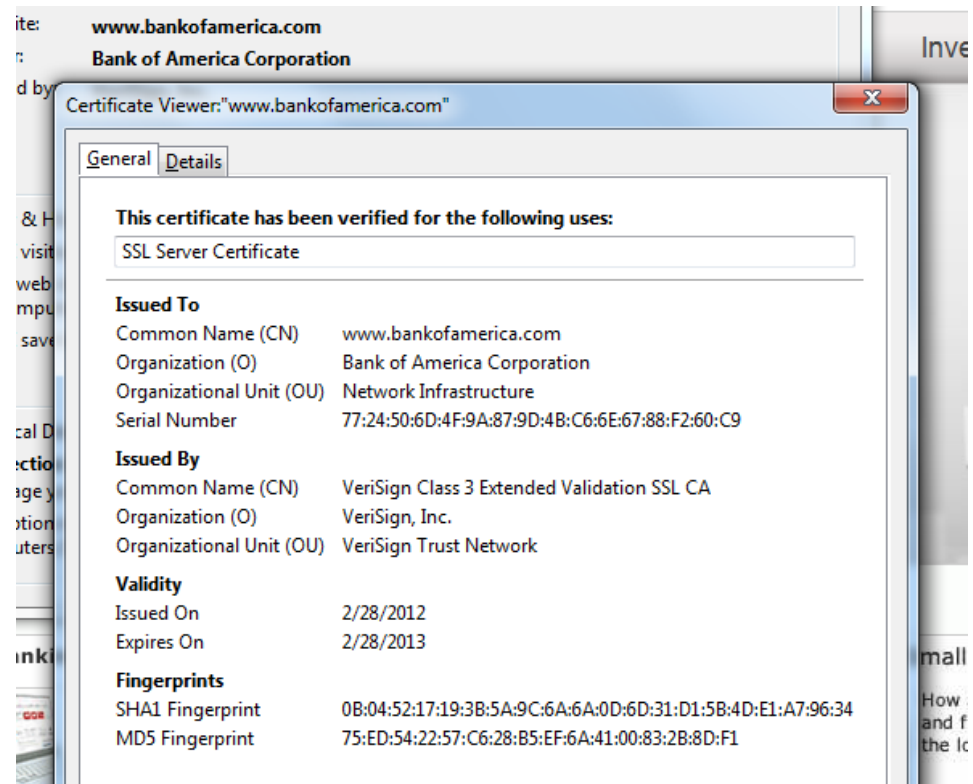 - Fairly transparent to the app

HTTPS Connection (SSL/TLS)

- 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)

Browser   **Amazon**

SYN

SYN ACK

ACK

Hello. I support
(TLS+RSA+AES128+SHA1) or
(SSL+RSA+3DES+MD5) or …

Let's use
TLS+RSA+AES128+SHA1

Here's my cert

~1 KB of data

Inside the Server's Certificate

- Name associated with cert (e.g., Bank of America)

Inside the Server's Certificate

- Name associated with cert (e.g., Bank of America)
- BoA'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 …
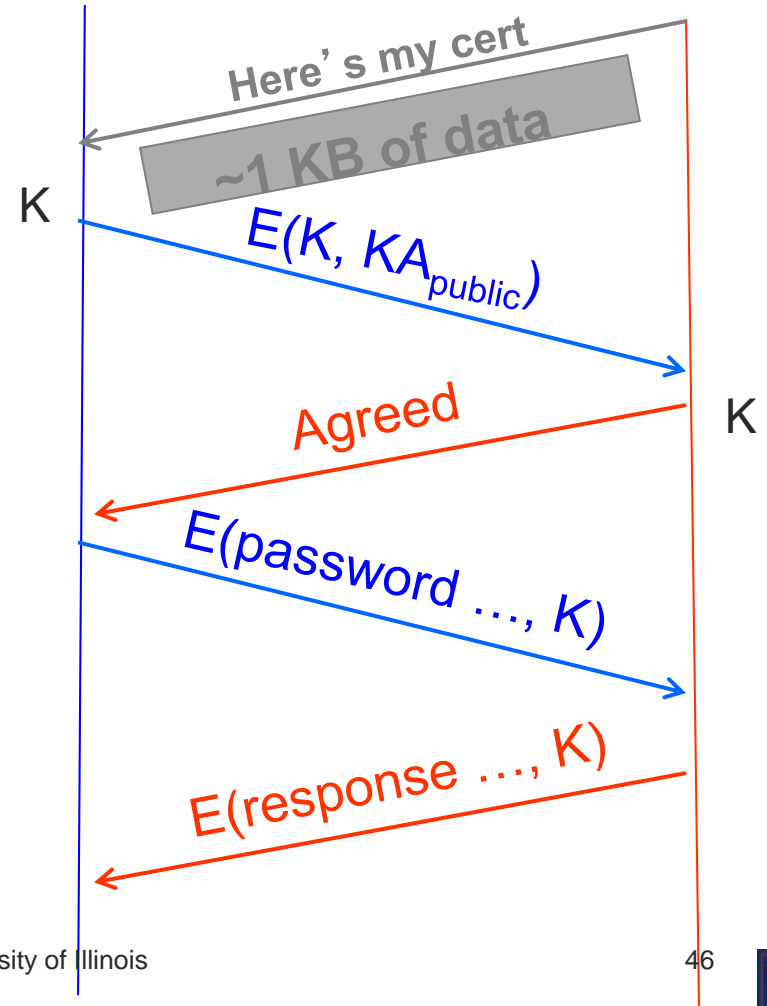 - … assuming signatory is trustworthy!

HTTPS Connection (SSL/TLS)

- 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

Browser   **Amazon**

Here's my cert

~1 KB of data

K

$E(K, KA_{public})$

K

Agreed

$E(password \dots, K)$

K

$E(response \dots, K)$

Solutions for basic security requirements

- **Confidentiality**: Encryption

- **Integrity**: Cryptographic Hash

- **Authentication**: Decrypting nonce

- **Provenance**: Digital signature
  - **Human-level provenance**: PKI

- **Availability**: ?

Solutions for basic security requirements

- **Confidentiality**: Encr
- **Integrity**: Cryptograp
- **Authentication**: Dec
- **Provenance**: Digital
  - **Human-level prover**

- **Availability**: ?

Crypto lets us convert "messy failures" into "clean failures" [Dave Clark]

e.g., authentication failure becomes connection drop

OK, so what about availability?

Protecting Availability

© CS 438 Staff - University of Illinois

Threats

Most Significant Operational Threats

Legend:
- DDoS Attacks Toward Customers
- Infrastructure Outages Due to Failures
- Botted/Compromised Hosts on Network
- DDoS Attacks Toward Services
- Infrastructure Outages Due to DDoS Attacks
- DDoS Attacks Toward Infrastructure
- New Vulnerabilities
- Zero-Day Exploits
- Undercapacity for Bandwidth
- Other

[Arbor Networks Security Report 2010]
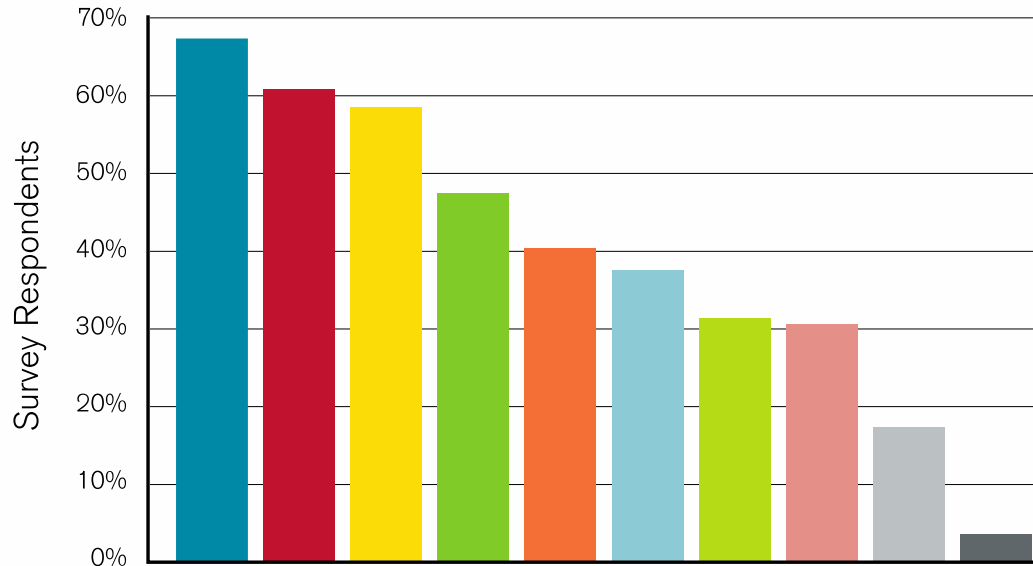
Figure 7

Source: Arbor Networks, Inc.

Threats to Availability

- Infrastructure compromise (e.g. BGP, DNS):
  - Attack removes service from operation
  - Design protocols to have limited Byzantine vulnerability
  - Prevent outsiders from posing as infrastructure (crypto)

- Denial-of-Service Attacks
  - Attack consumes service resources so legitimate users can't get any
  - What are they?
  - How can we defend against them?

Infrastructure compromise Example #1: BGP

- Any router can advertise any IP prefix to any other router
  - ○ Or, advertise fake path through attacker to legitimate owner
  - ○ Or, advertise anything …
- Once advertisement attracts incoming traffic…
  - ○ Just drop it ("black hole"): denial of service attack
  - ○ Eavesdrop
  - ○ Impersonate real destination
  - ○ Send spam using someone else's IPs; then disappear…

Infrastructure compromise Example #1: BGP

- No general-purpose automatic way of detecting suspicious or incorrect advertisements in BGP

- Eventual solution: secure variants of BGP incorporate cryptographic signatures
  - Doesn't fix all traffic attraction attacks

Infrastructure compromise Example #2: DNS

- What security issues does the design & operation of the Domain Name System raise?
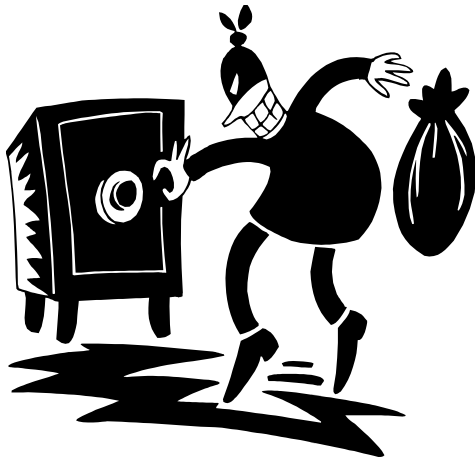
16 bits	16 bits
Identification	Flags
# Questions	# Answer RRs
# Authority RRs	# Additional RRs
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

DNS attack summary

- DNS currently lacks authentication
  - Can't tell if reply comes from the correct source
  - Can't tell if correct source tells the truth
  - Malicious source can insert extra (mis)information
  - Malicious bystander can spoof (mis)information
  - Playing with caching lifetimes adds extra power to attacks

- More importantly, example of how security was largely ignored in the original design of the Internet

Denial of Service (DoS)

- Attacker prevents legitimate users from using something (network, server)

- Increased workload
  - The overloaded component responds slowly or not at all to legitimate requests.

- Consider the following...

Denial of Service

This slide is intended to demonstrate the concept of denial of service by placing useful information in the middle of a paragraph of drivel. If you've gotten this far, you may want to jump to the middle or scan around, but you're unlikely to find the important part before I skip to the next slide, simulating the fact that the garbage doesn't stop coming in a denial of service attack. In fact, in some cases you may get nothing but garbage. If you recall how IP datagram services work, for example, then you can reason about the likelihood of legitimate requests spaced by increasingly long TCP timeouts finding an empty queue slot in a router fed from another input by a continuous stream of bogus packets. When a slot opens up, it is quickly grabbed by the next bogus packets and rarely available for the real thing.

Denial of Service

- So did you ge~~t the~~ point?

read

- During a denial of service attack, do servers continue to receive client requests?

Denial of Service

This slide is intended to demonstrate the concept of denial of service by placing useful information in the middle of a paragraph of drivel. If you've gotten this far, you may want to jump to the middle or scan around, but you're unlikely to find the important part before I skip to the next slide, simulating the fact that the garbage doesn't stop coming in a denial of service attack. In fact, in some cases you may get nothing but garbage. If you recall how IP datagram services work, for example, then you can reason about the likelihood of legitimate requests spaced by increasingly long TCP timeouts finding an empty queue slot in a router fed from another input by a continuous stream of bogus packets. When a slot opens up, it is quickly grabbed by the next bogus packets and rarely available for the real thing.

Denial of Service (DoS)

- Attacker prevents legitimate users from using something (network, server)
- Motives?
  - Retaliation
  - Extortion (e.g., betting sites just before big matches)
  - Commercial advantage (disable your competitor)
  - Cripple defenses (e.g., firewall) to enable broader attack

Denial of Service (DoS)

- Often done via some form of flooding

- Can be done at different semantic levels
  - Network
    - Clog a link or router with a huge rate of packets
  - Transport
    - Overwhelm victim's ability to handle connections
  - Application
    - Overwhelm victim's ability to handle requests

How Does It Traditionally Work?

- Example: IP spoofing
  - Remember the TCP SYN segment?
    - Client sends SYN to server
    - Server reserves queue entry
    - Server sends back SYN
  - Server sends back SYN to where?
    - Client IP must be included in first SYN
    - What if the client lies?
      "Hi, please contact Riccardo's computer."

IP Spoofing

- Pretending to be Riccardo's computer
  - It responds
  - Queue resources freed up
  - Not terribly effective
- Instead, pick a computer that probably won't respond
  - A random number works nicely
  - Return SYN gets lost
  - Server waits 75 seconds before freeing queue entry
- What are the key elements?

Key Elements of Denial of Service Attacks

- Expansion in required work
  - Easy for me, hard for you
  - In spoofing,
    - Creating and sending a SYN: a few microseconds
    - Timing out a queue entry: 75 seconds
- Protocols that admit starvation
  - IP routers
    - Drop datagrams when output buffer full
    - Independent of source input
  - Result is that clients can be starved
  - Painfully slow even without starvation

What's New about the Recent Attacks?

- Expansion factor allows attack by a few or one
- Alternative
  - Attack by many
  - Requires many resources distributed across Internet
  - Benefits from expansion, too
- Attack software
  - Probably installed indirectly
  - Apparently resident for some time
  - No significant trail remains

What Changed to Make These Attacks Possible?

- **Change of Internet character**
  - Many more machines
  - Many more naive users
  - Much more complex software
- **Old Internet**
  - Very little interpretation of data (e.g., e-mail)
  - Operating system bugs relative secure (by obscurity)
  - Security-savvy administrator required (so security breaches detected and repaired)

Denial of Service Attacks in the New Internet

- **New Internet**
  - Point-and-click network installation
  - Very broad interfaces
  - Even transparent code encapsulation
  - And self-installing "plug-ins"!
  - Public source operating systems, too
  - (and frustrated security gurus writing tools to eliminate any remaining obscurity)
- **Example of attack:**
  - Abuse bug in Internet Explorer to install flashing window telling owner to download patch
  - What else might someone install instead?

IP Spoofing Countermeasures

- IP spoofing
  - Abuses TCP connection queue time expansion
  - Considered unsolvable for quite some time
  - Solved by ingenious use of cryptography
- Solution
  - Return one-use key with response SYN segment
  - Reserve no queue resources
  - ACK to second SYN (third step of setup) must return the key
  - IP spoofing never sends such an ACK

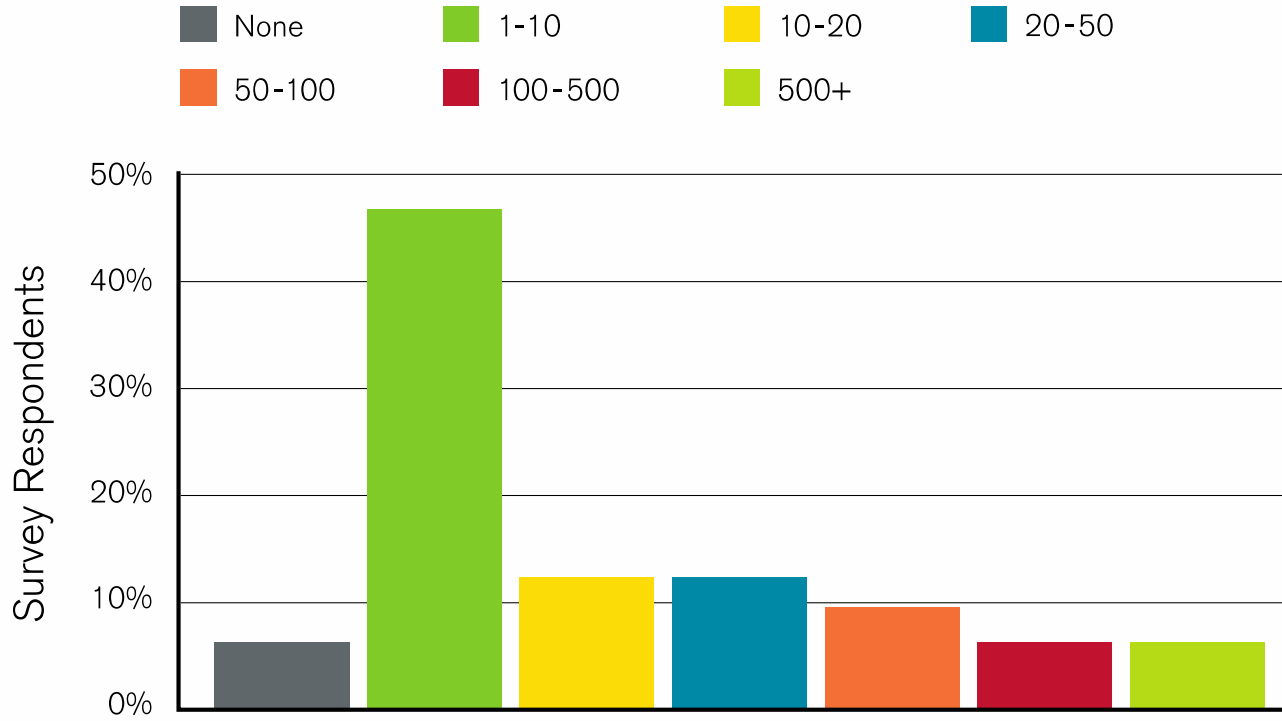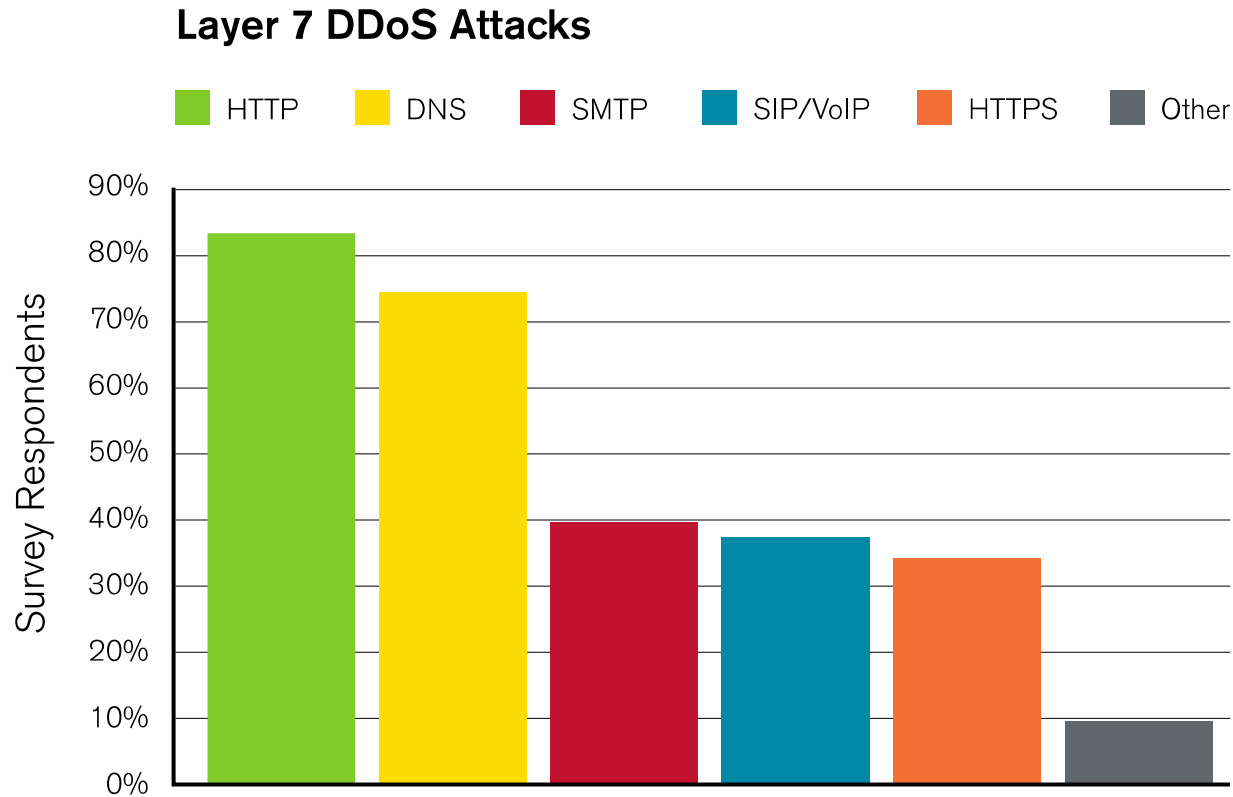# Denial of Service (DoS)

Average Number of DDoS Attacks per Month

Legend:
- None (gray)
- 1-10 (green)
- 10-20 (yellow)
- 20-50 (teal)
- 50-100 (orange)
- 100-500 (dark red)
- 500+ (light green)

Figure 15
Source: Arbor Networks, Inc.

Denial of Service (DoS)

Figure 8
Source: Arbor Networks, Inc.

DoS: Network Flooding

- Goal is to clog network link(s) leading to victim
  - Either fill the link, or overwhelm their routers
  - Users can't access victim server due to congestion
- Attacker sends traffic to victim as fast as possible
  - It will often use (many) spoofed source addresses

DoS: Network Flooding

- Using multiple hosts (slaves, or zombies) yields a Distributed Denial-of-Service attack, aka DDoS

- Traffic can be varied (sources, destinations, ports, length) so no simple filter matches it

- If attacker has enough slaves, often doesn't need to spoof - victim can't shut them down anyway! :-(

Distributed Denial-of-Service (DDoS)

src = random
dst = victim

Slave 1

Slave 2

Slave 3

Slave 4

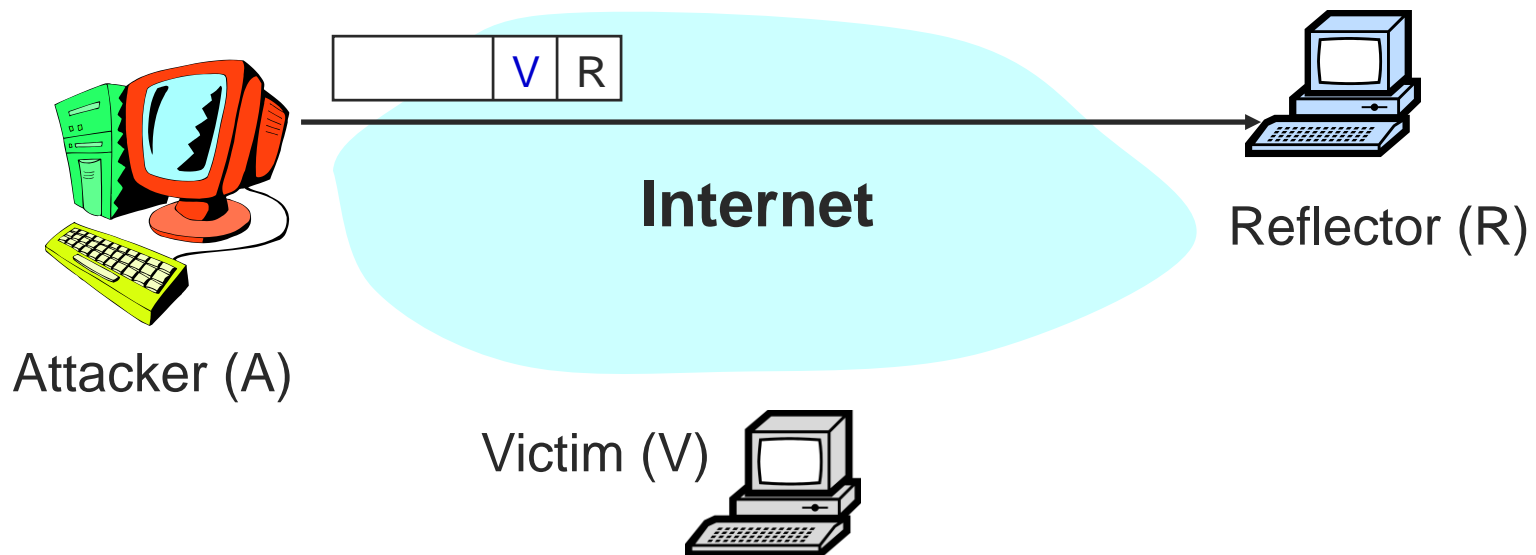Master
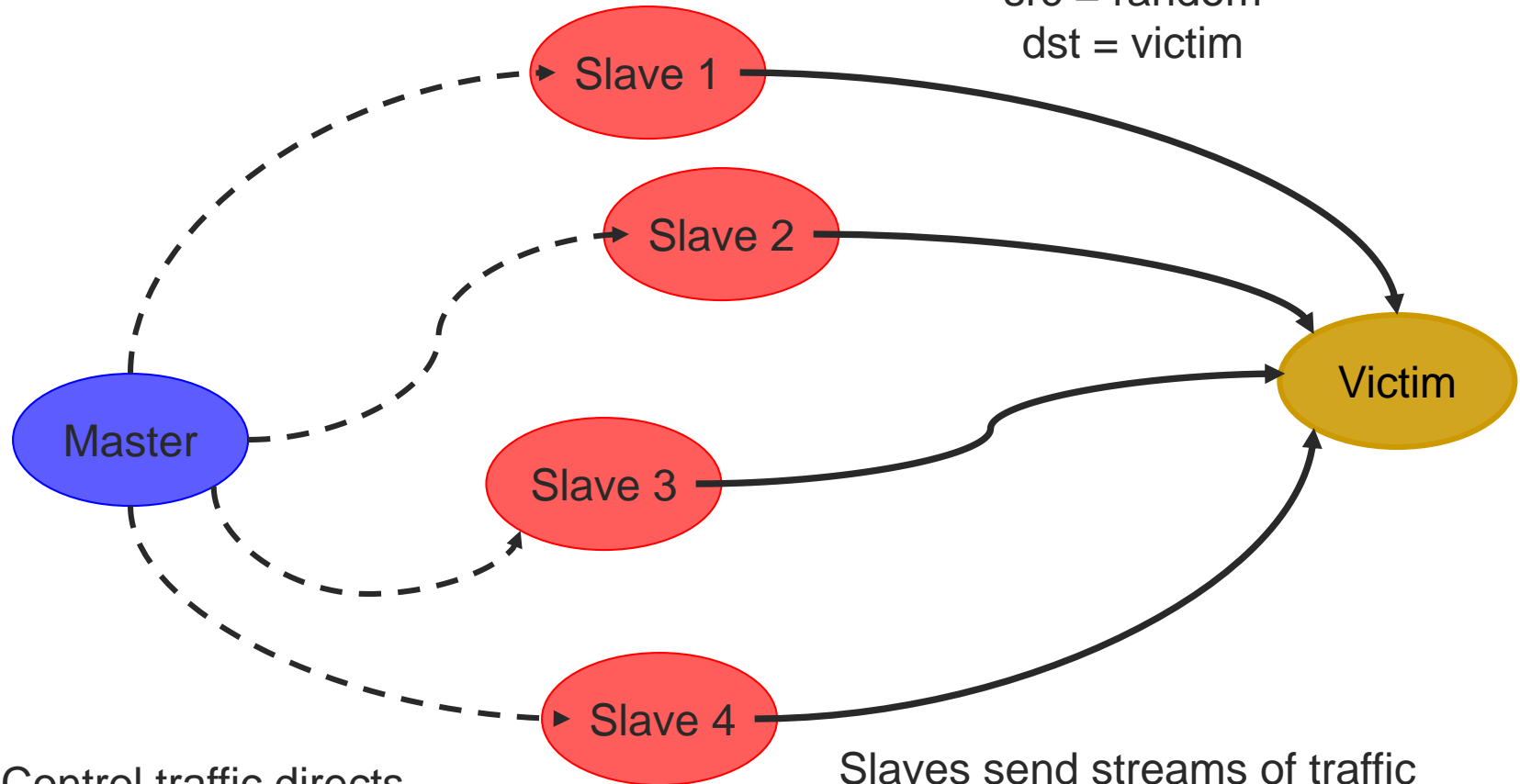
Victim

Control traffic directs slaves at victim

Slaves send streams of traffic (perhaps spoofed) to victim

© CS 438 Staff - University of Illinois

Very Nasty DoS Attack: Reflectors
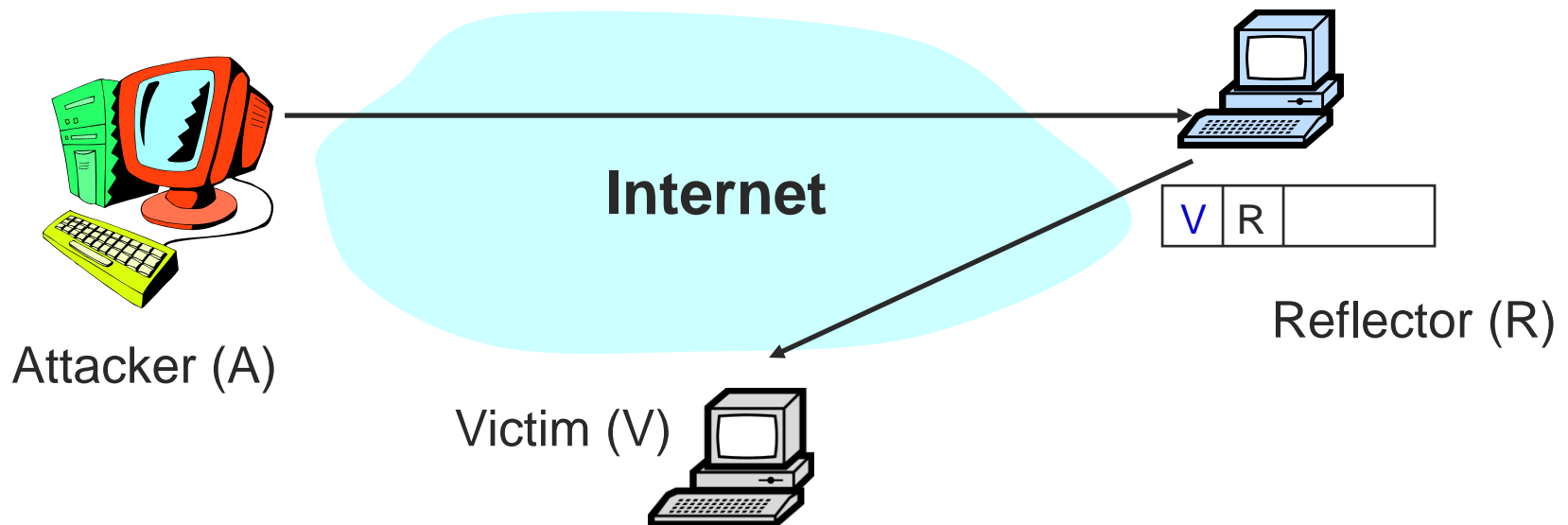
- **Reflection**
  - Cause one non-compromised host to help flood another
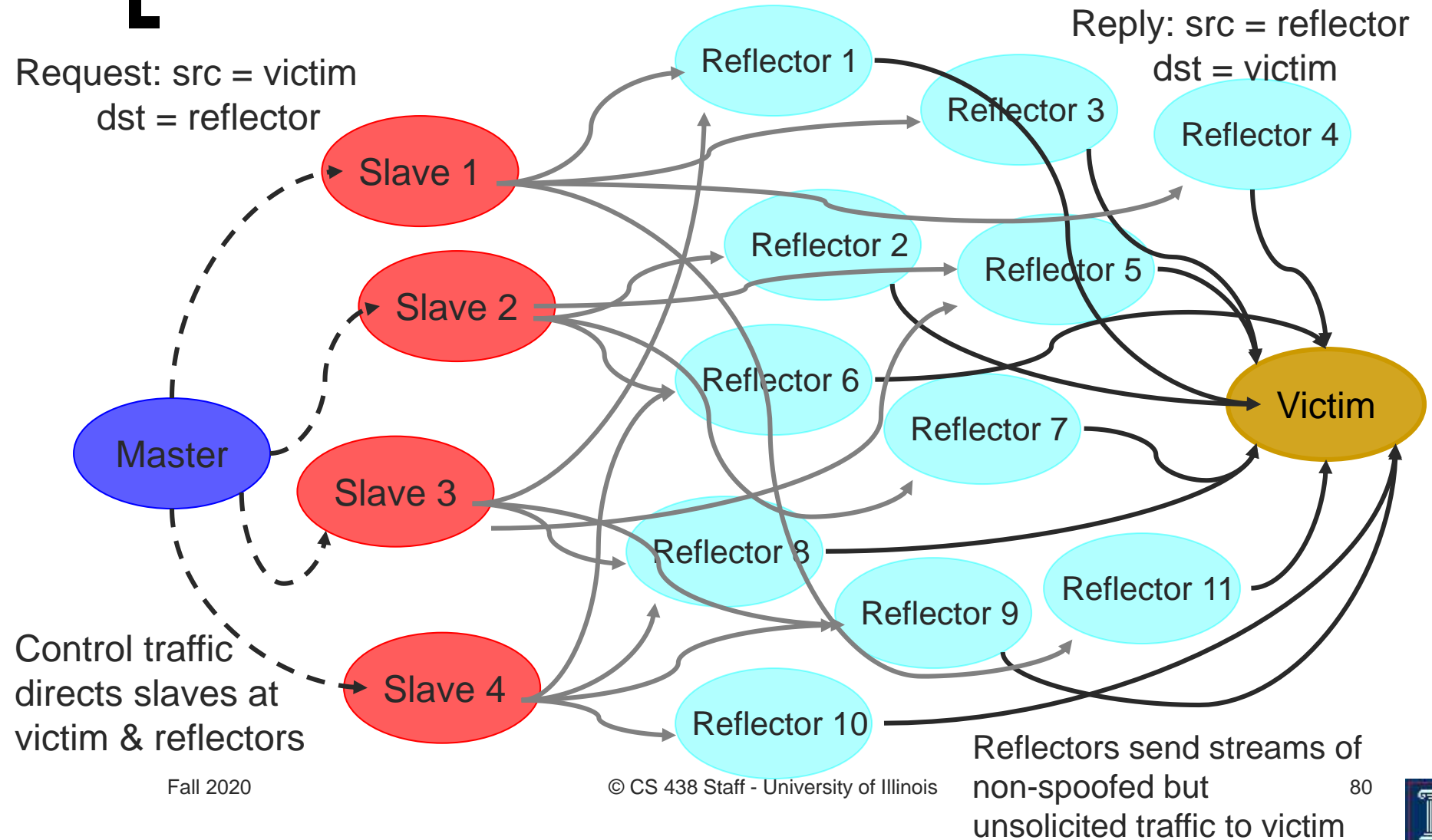 - e.g., host A sends DNS request or TCP SYN with source V to server R.

Very Nasty DoS Attack: Reflectors

- **Reflection**
  - Cause one non-compromised host to help flood another
  - e.g., host A sends DNS request or TCP SYN with source V to server R.

Internet

Attacker (A)

V	R	

Reflector (R)

Victim (V)

Diffuse DDoS: Reflector Attack

Request: src = victim
dst = reflector

Reply: src = reflector
dst = victim

Reflector 1
Reflector 3
Reflector 4
Reflector 2
Reflector 5
Reflector 6
Reflector 7
Reflector 8
Reflector 9
Reflector 11
Reflector 10

Slave 1
Slave 2
Slave 3
Slave 4

Master

Victim

Control traffic directs slaves at victim & reflectors

Reflectors send streams of non-spoofed but unsolicited traffic to victim

Defending Against Network Flooding

- How do we defend against such floods?
- Answer: we don't! (not completely.)
  - Big problem today!
- Techniques exist to trace spoofed traffic back to origins
  - Not useful in face of a large attack
- Techniques exist to filter traffic
  - A well-designed flooding stream defies stateless filtering

Defending Against Network Flooding

- **Best solutions to date**
  - Overprovision - have enough raw capacity that it's hard to flood your links
    - Largest confirmed botnet to date: 1.5 million hosts
    - Floods seen to date: as high as 100 Gbps
  - Distribute your services - force attacker to flood many points
    - e.g., the root name servers

Proposed Solutions

- **Network-level attacks**
  - Capabilities: don't let flows send without permission
  - Shut-up message

- **Application-level attacks**
  - Proof-of-work
  - Ask clients to send more

We solved security!

Hooray!

…or not…It is a Big Bad World Out There…

Host Compromise

- Tricking a host into executing on your behalf

- Can consider what is attacked (server or client) and the semantic level at which it is attacked

- Attacks on servers: client sends subversive requests

  - Happens at attacker's choosing
  - Some hosts are servers unknowingly!

Host Compromise

- Attacks on clients: server (attacker) waits for client to connect, sends it a subversive reply
  - e.g., "drive-by" spyware
  - e.g., 2006 study found 15% of popular P2P files infected by one of 52 different viruses
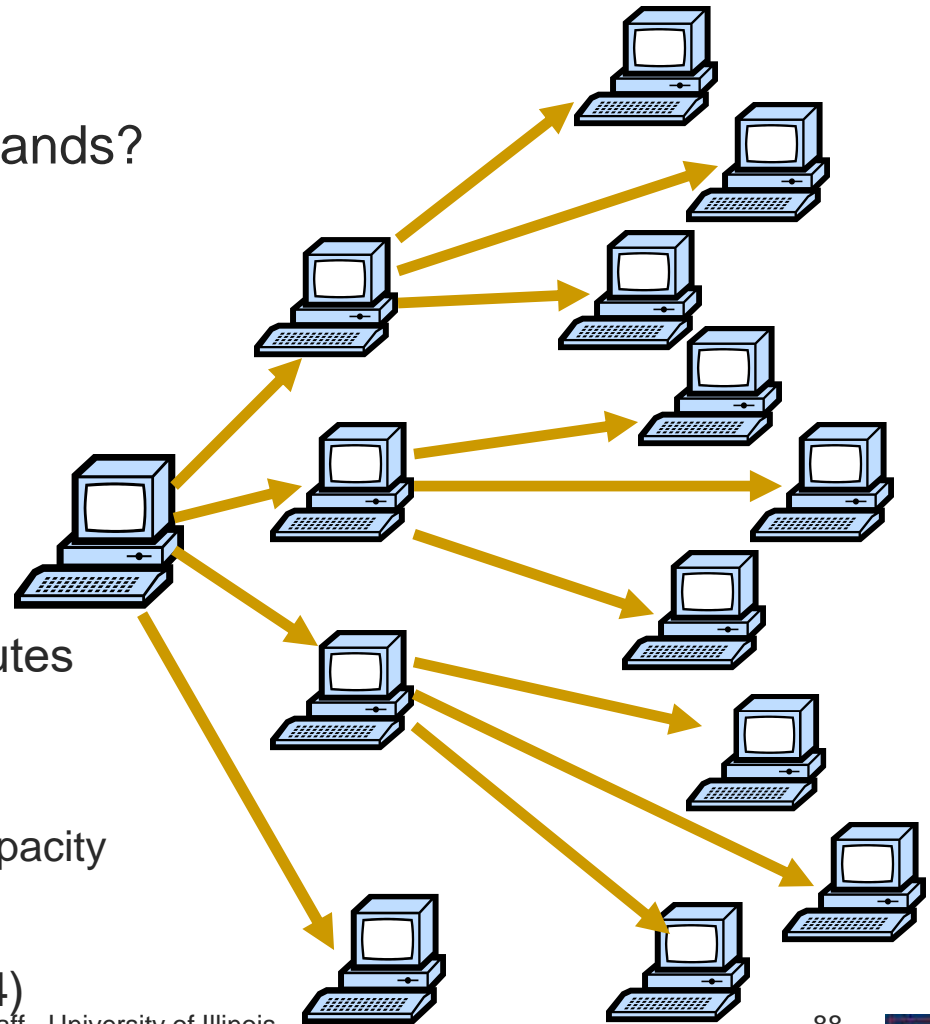
Automated Compromise: Worms

- When attacker compromises a host, they can instruct it to do whatever they want

- Instructing it to find more vulnerable hosts to repeat the process creates a worm: a program that self-replicates across a network

  - Often spread by picking 32-bit Internet addresses at random to probe …

  - … but this isn't fundamental

- As the worm repeatedly replicates, it grows exponentially fast because each copy of the worm works in parallel to find more victims

Worms: Exponentially Fast .... and Big

- **Morris Worm (1988): thousands?**
- **Code Red 1 (2001)**
  - 369K hosts in 10 hours
- **Blaster (2003)**
  - 9M hosts in 9 days
  - 25M hosts total
- **Slammer (2003)**
  - 75K hosts … in < 10 minutes
  - Peak scanning rate: 55M addresses/sec
    - Limited by Internet's capacity
- **Theoretical worms**
  - 1M hosts in 1.3 sec (2004)
  - $50B+ damage (2004)

Automated Compromise: Bots

- Big worms are flashy but rare …

- With the commercialization of malware, tool of choice has shifted to the less noisy, more directly controlled botnets

- When host is (automatically) compromised, don't continue propagation
  - Instead install a command and control platform (a bot)

- Now can monetize malware: sell access to bots
  - Spamming, phishing web sites, flooding attacks
  - "Crook's Google Desktop": sell capability of searching the contents of 100,000s of hosts