

# **CSE 123b**

# **Communications Software**

**Spring 2003**

**Lecture 15: Network Security I**

Stefan Savage

# Overview

---

- What is network security?
- Communications channel vulnerabilities
  - ♦ End-to-end cryptography
- System software vulnerabilities
  - ♦ Perimeter defenses
- Protocol vulnerabilities
  - ♦ Deliberate Misinformation

# Network Security?

---

- What properties do we want?
  - ◆ Confidentiality, Integrity, Authenticity
  - ◆ Access control
  - ◆ Availability
  - ◆ Non-repudiation?
  - ◆ Consistency?
  - ◆ Privacy?
- What is challenging about the network environment?
  - ◆ Exposure/sharing
  - ◆ Anonymity
  - ◆ Fragility

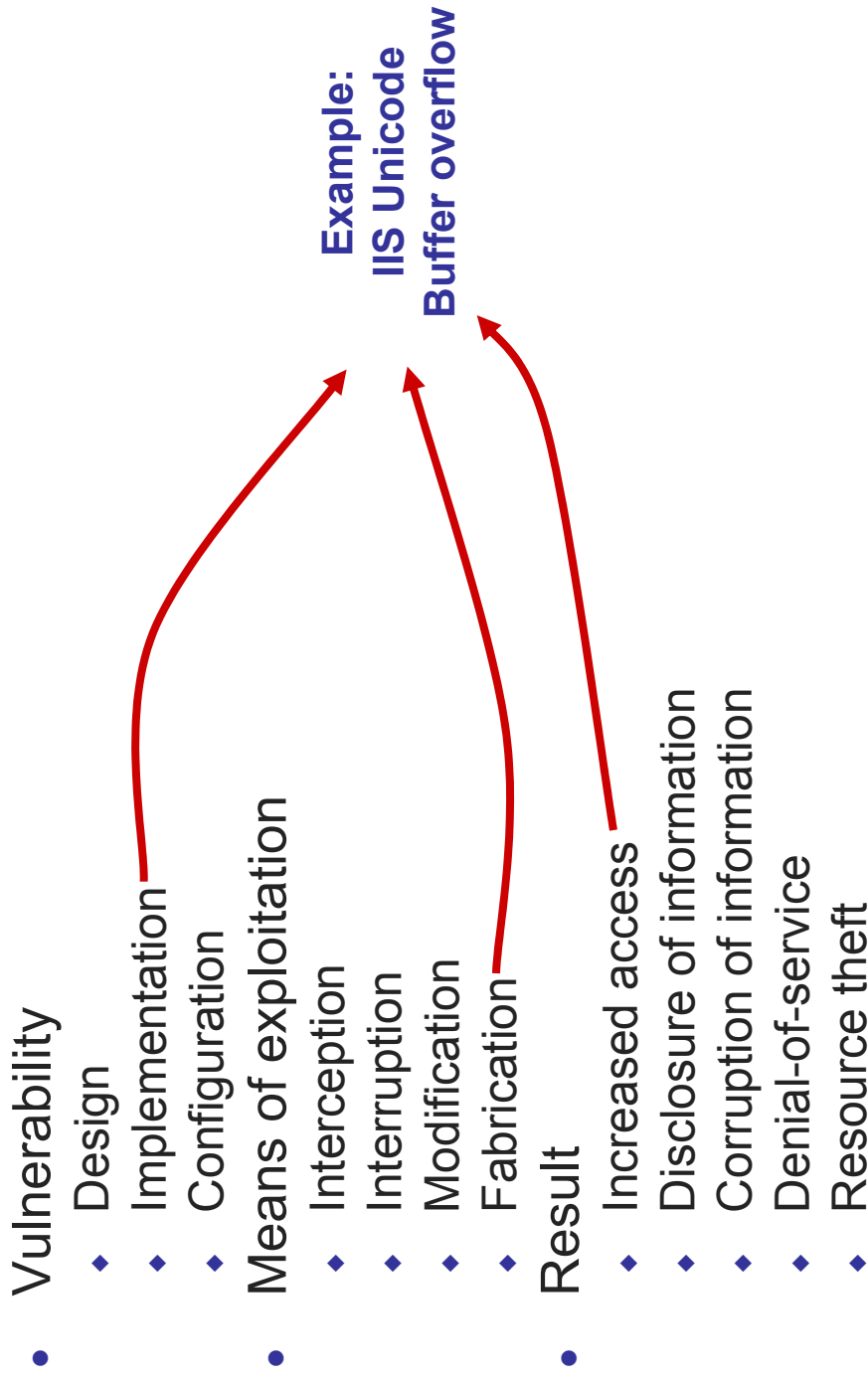
# Approaches at 10,000 ft

---

- Physical security
  - ◆ Tackle the problem of sharing directly
- “Security through obscurity”
  - ◆ Hope no-one will find out what you’re doing!
- Throw math at the problem
  - ◆ Cryptography
- Why is security difficult?
  - ◆ It’s a negative goal: can you be sure there are no flaws?
  - ◆ Often assumptions turn out to be invalid

# Taxonomy of attacks

---



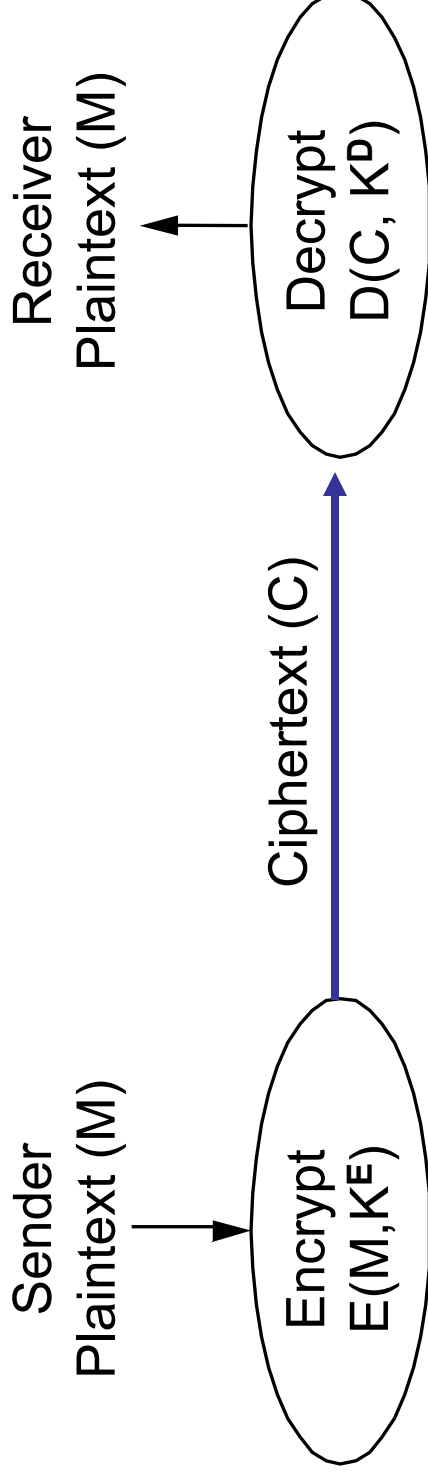
# Communications channel vulnerabilities

---

- **Confidentiality**
  - ◆ Attacker can intercept messages (passwords, data)
  - ◆ Easy on local network; harder at a distance
- **Integrity**
  - ◆ Attacker can change messages unbeknownst to sender/receiver
  - ◆ Marginally harder attack – must intercept or stop forwarding of legitimate messages
- **Authenticity**
  - ◆ Attacker can “pretend” to be a user illegitimately
  - ◆ Easy

# Basic Encryption for Confidentiality

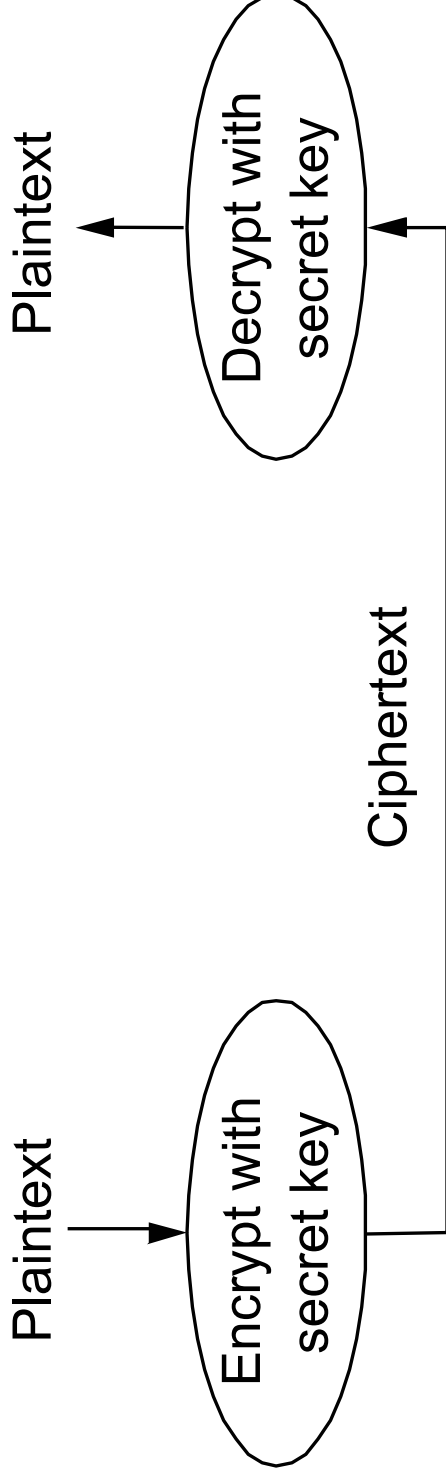
---



- Cryptographer chooses functions  $E$ ,  $D$  and keys  $K^E$ ,  $K^D$ 
  - ♦ Solving  $D(C, x) = M$  should be hard without  $x$
- Cryptanalyst try to “break” the system
  - ♦ Depends on what is known:  $E$  and  $D$ ,  $M$  and  $C$ ?

# Symmetric Key Functions (DES, IDEA, AES)

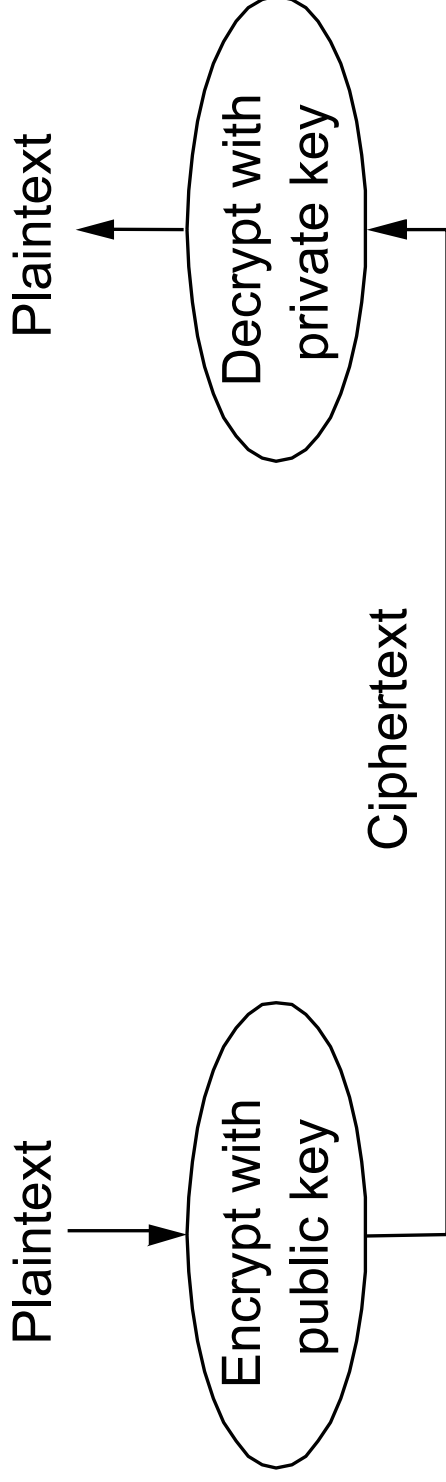
---



- $K^E, K^D = K; E(M, K) = \{M\}^K, D(\{M\}^K, K) = M$
- Key must be communicated to both parties, but must be secret to everyone else (key distribution problem)
- Encryption/decryption fast and have equivalent cost
- Also called secret-key or shared-key cryptography

# Asymmetric Key Functions (RSA)

---



- $K^E =$  secret key (SK)  $K^D =$  public key (PK)
  - ♦  $E(M, SK) = \{M\}^{SK}$ ,  $D(\{M\}^{SK}, PK) = M$
  - ♦  $E(M, PK) = \{M\}^{PK}$ ,  $D(\{M\}^{PK}, SK) = M$
- DES 100 times faster than RSA in software
  - ♦ Typically, PK/SK used to exchange symmetric key, which is used for the conversation
  - ♦ PK can be exchanged “in the clear” (issues?)

# Integrity (MD5, SHA)

---

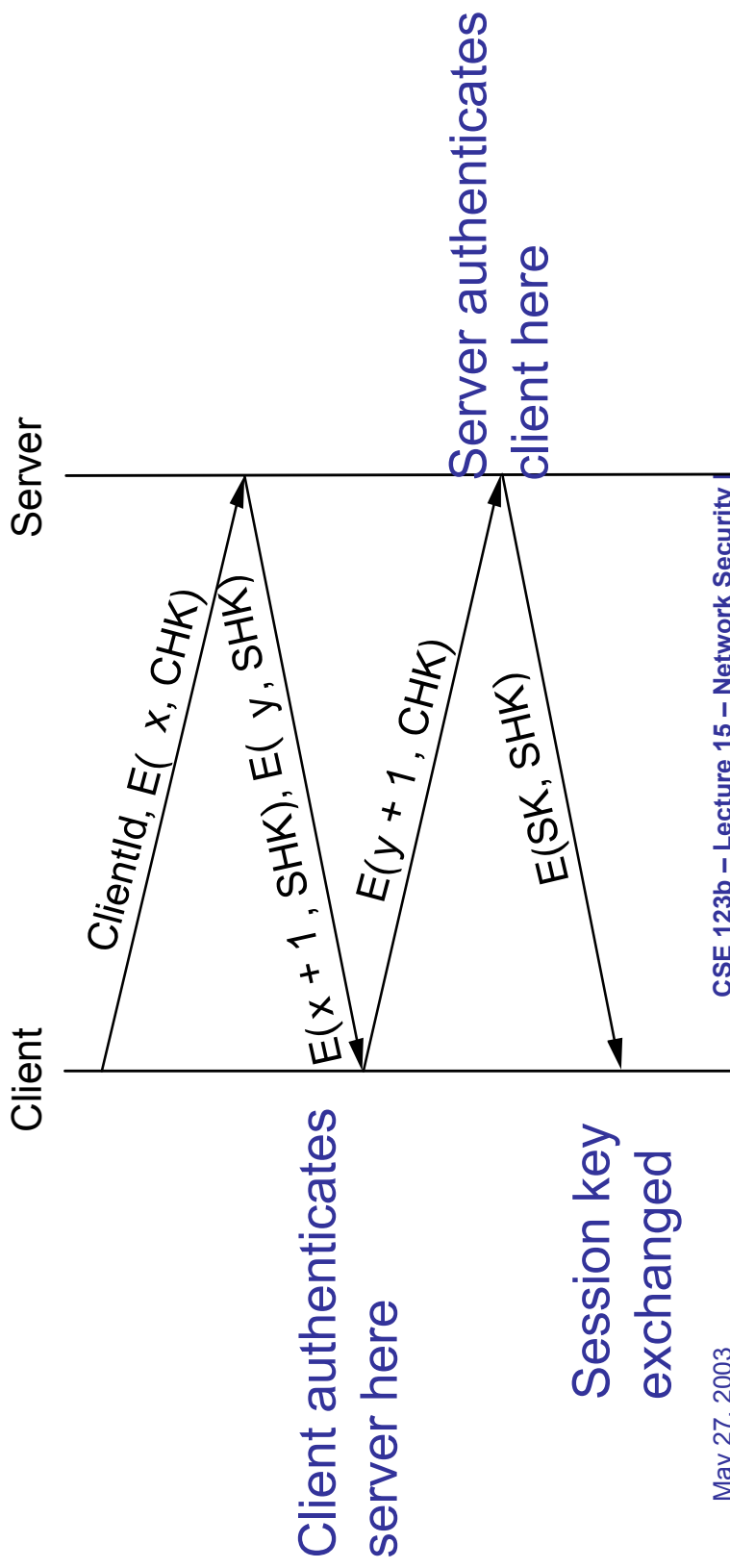
- Verify that a message has not been modified
  - ♦ Much stronger than checksum (difference?)
- Message digest/ characteristic function/ one-way hash:
  - ♦  $H(M) = h$
  - ♦  $h, H \neq M$  (inversion resistance) [also called one-way]
  - ♦  $M \neq M'$ , s.t.  $H(M) = H(M')$  (collision resistance)
  - ♦ Additional mechanism to prevent attacker from also modifying hash
    - » encrypt  $h$ , or
    - »  $h = H(M, K)$ ,  $K$  is a secret key known by both sender and receiver

# Authenticity

## Symmetric (secret) keys

---

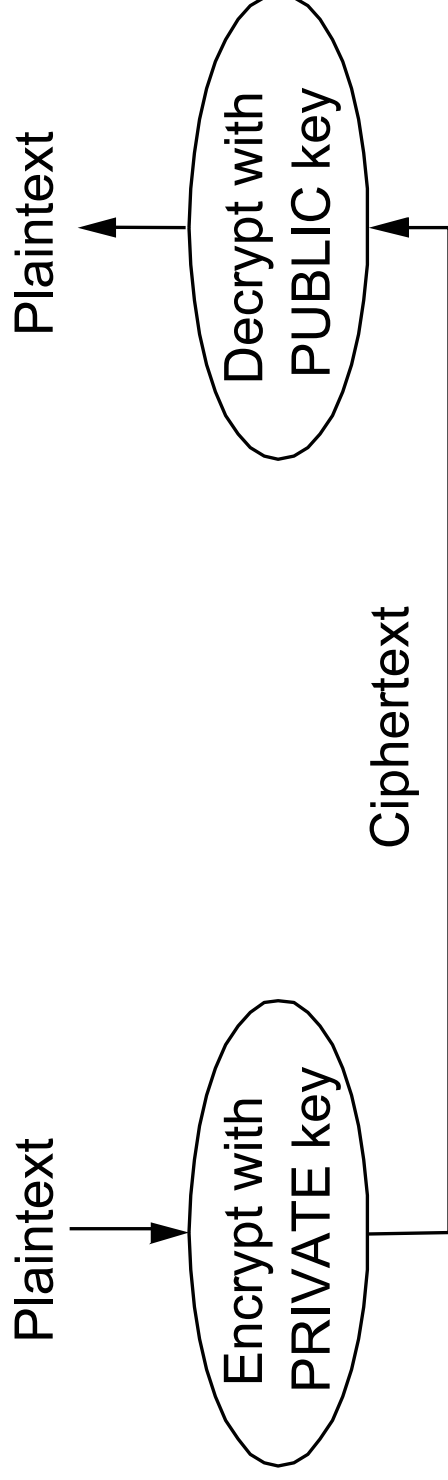
- Three-way handshake for mutual authentication
  - ♦ Client and server share secrets, e.g., login password



# Authenticity

## Asymmetric (public) keys

---



- Notice that we reversed the role of the keys (and the math just works out) so only one party can send the message but anyone can check it's authenticity

# Digital signatures

---

- Encryption can be expensive, e.g., RSA 1Kbps
- To speed up, let's just encrypt the message digest/hash instead!
- Absolutely critical that hash is “cryptographically strong”
  - ◆ Inversion resistance, collision resistance
  - ◆ Related to size of hash

# Example: SSL

---

- Transport layer secure channel
- Connection setup
  - ◆ Negotiate encryption algorithm
  - ◆ Server provides SSL certificate
    - » Certification Authority (CA), CA signature, principal, principals public key and timeout
  - ◆ Client validates certificate (digital signature) using well-known public-key for CA ,
  - ◆ If valid, can use principal's public key to negotiate session key
- Symmetric session key used to encrypt channel
- Who is trying to establish trust with whom here?

# System-level vulnerabilities

---

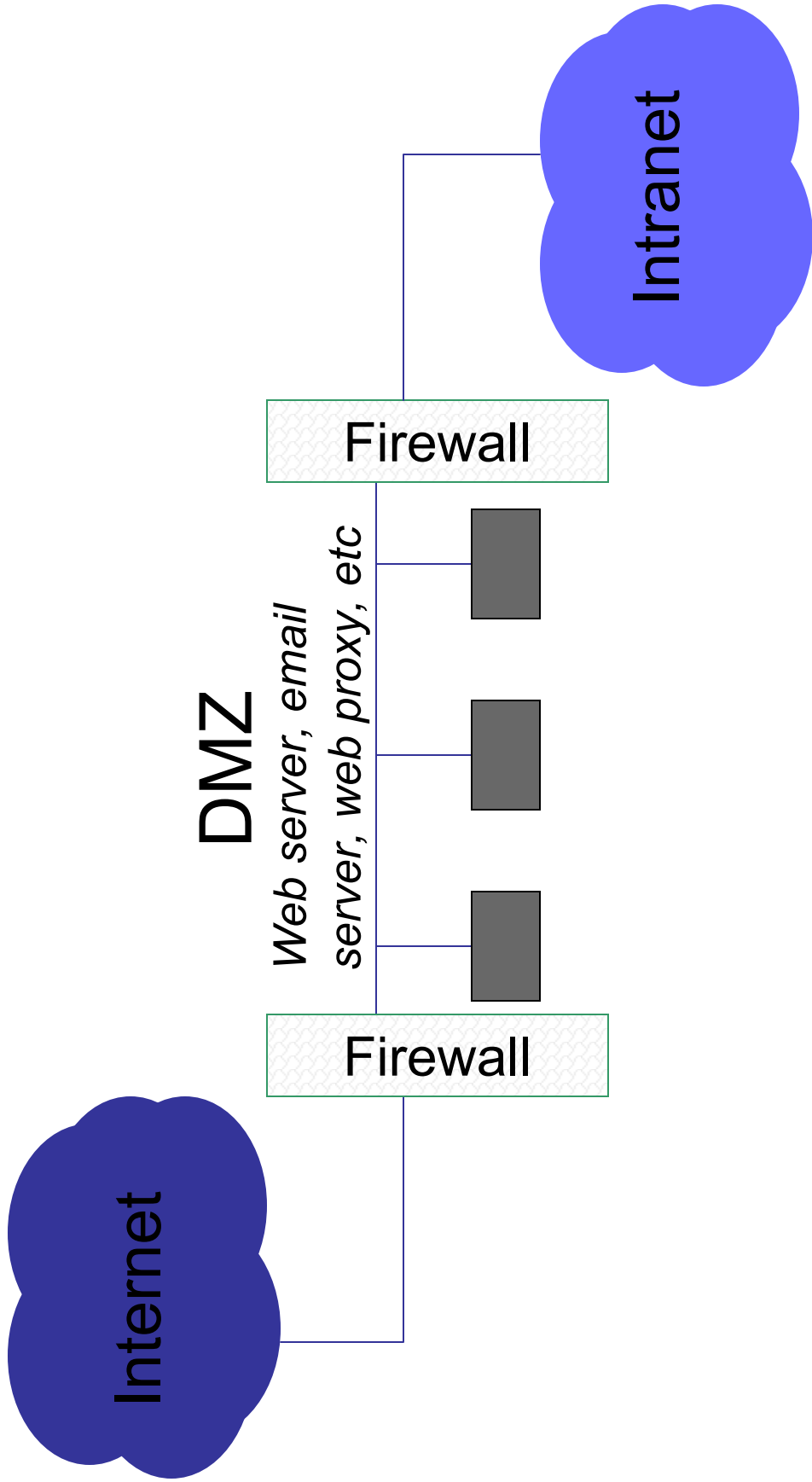
- How often is security break caused by breaking crypto?
  - ♦ Why/where is strength/weakness of crypto important?
- Implementation bugs principal technical source of host compromises
  - ♦ Buffer overflow
  - ♦ Unchecked parameters
  - ♦ Randomness assumptions
  - ♦ Race condition
- Ideally: patch/fix all the hosts so no vulnerabilities can be exploited

# Perimeter defenses

---

- Key ideas:
  - ♦ Too hard to secure/patch/fix each individual system
  - ♦ Install “watchdog” system at perimeter of network to protect all hosts inside
  - ♦ Model: internal machines are trusted, external machines are untrusted
- **Network address translation**
  - ♦ Multiplex internal address space on small number of public IP addresses; internal hosts can’t be addressed directly from the outside
- **Firewalls**
  - ♦ Limit access to end hosts (only those hosts/services that must be made public can be accessed from the outside)
- **Intrusion detection systems**
  - ♦ Detect attempts to break into hosts or exploit system vulnerabilities

# Typical Firewall Topology



# Types of Firewalls

---

- **Proxy**
  - ◆ End host connects to proxy and asks it to perform actions on its behalf
    - » Policy determines if action is secure or insecure
  - ◆ Transport level relays (SOCKS)
    - » Ask proxy to create, accept TCP (or UDP) connection
    - » Cannot secure against insecure application
  - ◆ Application level relays (e.g. HTTP, FTP, telnet, etc.)
    - » Ask proxy to perform application action (e.g. HTTP Get, FTP transfer)
    - » Can use application action to determine security
  - ◆ Requires applications to be modified to use the proxy
  - ◆ Considered to be the most secure since it has most information to make decision

# Types of Firewalls

---

- **Packet filters**
  - ◆ Set of filters and associated actions that are used on a packet by packet basis
  - ◆ Filters specify fields, masks and values to match against packet contents, input and output interface
  - ◆ Actions are typically **forward** or **discard** (yes or no)
  - ◆ Such systems have difficulty with things like fragments and a variety of attacks
  - ◆ Typically a difficult balance between the access given and the ability to run applications
    - » E.g. FTP often needs inbound connections on arbitrary port numbers – either make it difficult to use FTP or limit its use

# Types of Firewalls

---

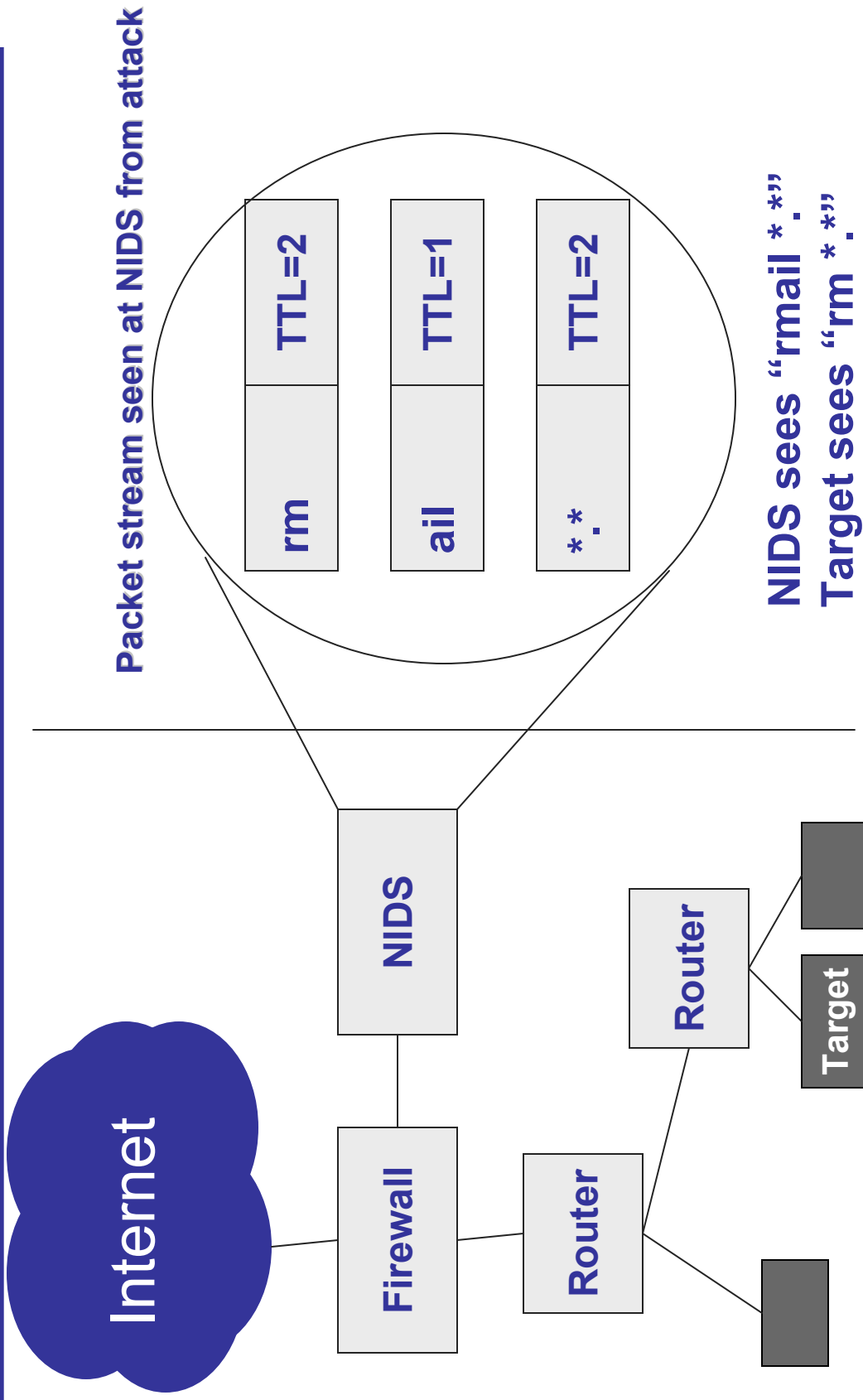
- **Stateful packet filters**
  - ◆ Allocate state for each flow (i.e. each TCP session)
  - ◆ Typically allow richer parsing of each packet (variable length fields, application headers, etc.)
  - ◆ Actions can include the addition of new rules and the creation of state to process future packets
    - » Often have to parse application payload to determine “intent” and determine security considerations
  - ◆ Rules can be based on packet contents and state created by past packets
  - ◆ Provides many of the security benefits of proxies but without having to modify applications

# Network Intrusion Detection Systems

---

- Deployed in similar manner to firewalls
  - ◆ Frequently not “in-line” (i.e. if IDS fails, traffic continues)
- Observe all packets and check for intrusion attempts
  - ◆ Signature detection (e.g. any HTTP packets with “rm -rf” them)
  - ◆ Anomaly detection (e.g. unusual sized requests to port 79)
- Issues
  - ◆ False negatives, False positives
  - ◆ Evading detection
  - ◆ Overhead (can be overwhelmed)
  - ◆ Still expensive to respond

# Example: Evading a NIDS



# Protecting against evasion

---

- **Traffic normalization**
  - ◆ Overwrite packets to make them consistent
  - ◆ E.g. all packets get same ttl
- **Active Mapping**
  - ◆ NIDS measures per destination characteristics and adjust analysis accordingly
  - ◆ E.g. how far each host is from NIDS

# Protocol vulnerabilities

---

- Even if two endpoints have authenticity, integrity, & confidentiality that doesn't mean they will behave
  - ♦ Where does trust work as a security mechanism?
- Examples
  - ♦ Routing protocols
  - ♦ TCP congestion control

# Routing attacks

---

- Problem: Attacker may advertise bogus routes
  - ◆ Claim to originate network/host
  - ◆ Intercept packets then re-route to true destination
  - ◆ May also cause denial-of-service
- Solutions
  - ◆ Policy about which routes you believe (don't accept routes for own network); have well-known neighbors
  - ◆ Authentication of routing protocol sessions
  - ◆ Open research problem to handle this problem efficiently...

# TCP Congestion Control with a Misbehaving Receiver [Savage+99]

---

- Simple Question
  - ◆ Can a TCP client influence how fast a TCP server sends it data?
- Simple Answer: Yes!
- Outline:
  - ◆ Why this matters
  - ◆ The attacks
  - ◆ Some countermeasures

# The tragedy of the commons

---

- Internet bandwidth is a shared resource
  - ◆ Stability depends on voluntary end-to-end congestion control
  - ◆ If an individual host has both the incentive and ability to *cheat* then the entire system fails
- TCP senders (i.e. content servers)
  - ◆ Clearly have ability to cheat (send too fast)
  - ◆ Not strong incentive to cheat; own the whole commons
    - » Few senders, each high volume, diverse receivers
- TCP receivers (i.e. Web browsers)
  - ◆ Clearly have incentive to *steal* bandwidth
  - ◆ Not obvious they have ability

# Sources of vulnerability

---

- ACKs *mean* things that they don't *prove*
  - ◆ I was sent in response to a data packet
  - ◆ That data packet has been received
  - ◆ I have received all the data up to X-1
  - ◆ I have (still) not yet received data X
- Sender assumes things that aren't necessarily true
  - ◆ At most one ACK generated per data packet
  - ◆ Every ACK acknowledges a full-sized packet

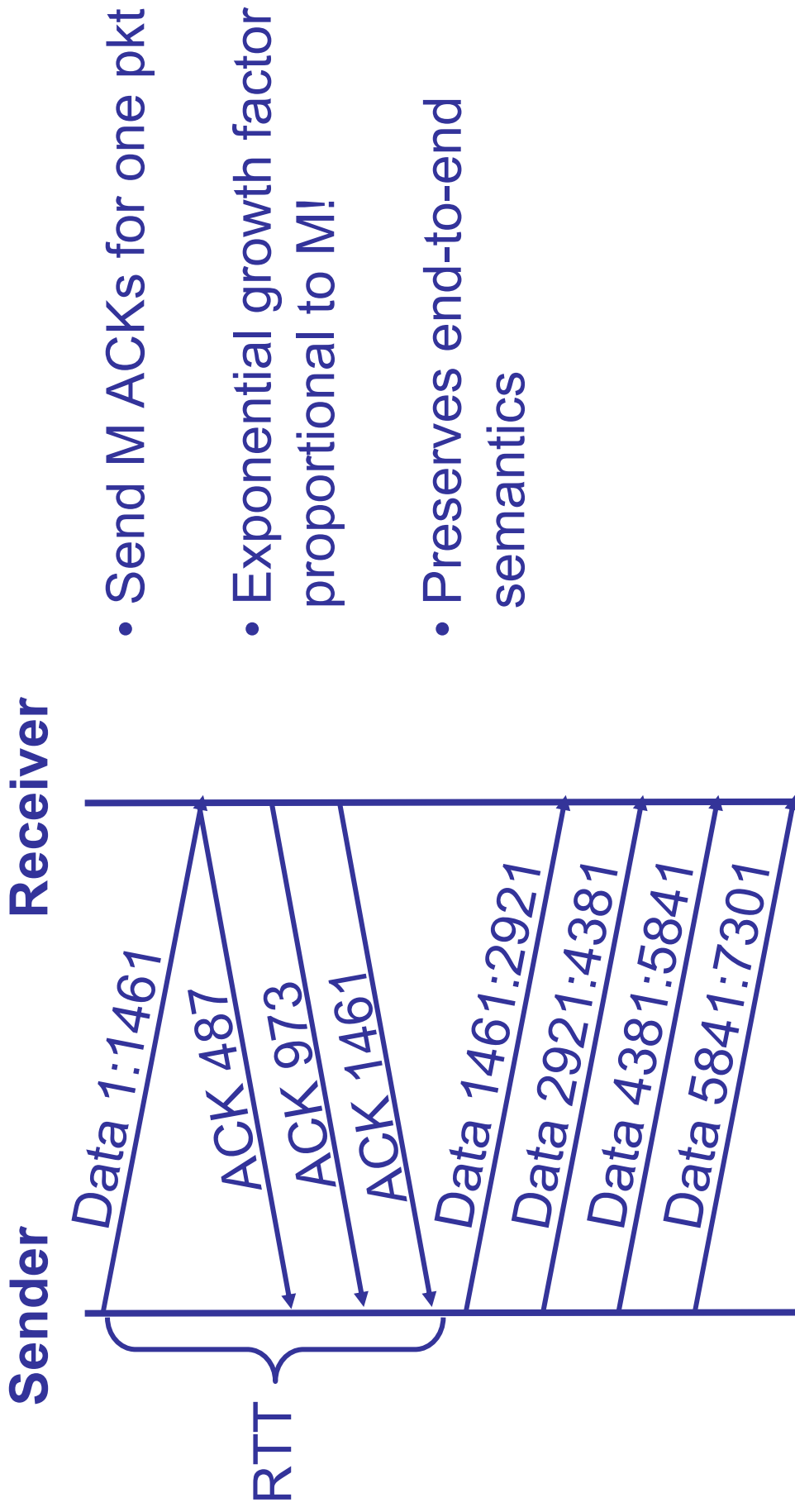
# Vulnerability 1: Bytes vs. Segments

---

- TCP: reliable byte stream w/ cumulative ACKs
- Cwnd limits unacknowledged data
- TCP begins a session in slow start:  
*During slow start, TCP increments cwnd by at most MSS bytes [one full sized packet] for each ACK received that acknowledges new data.*

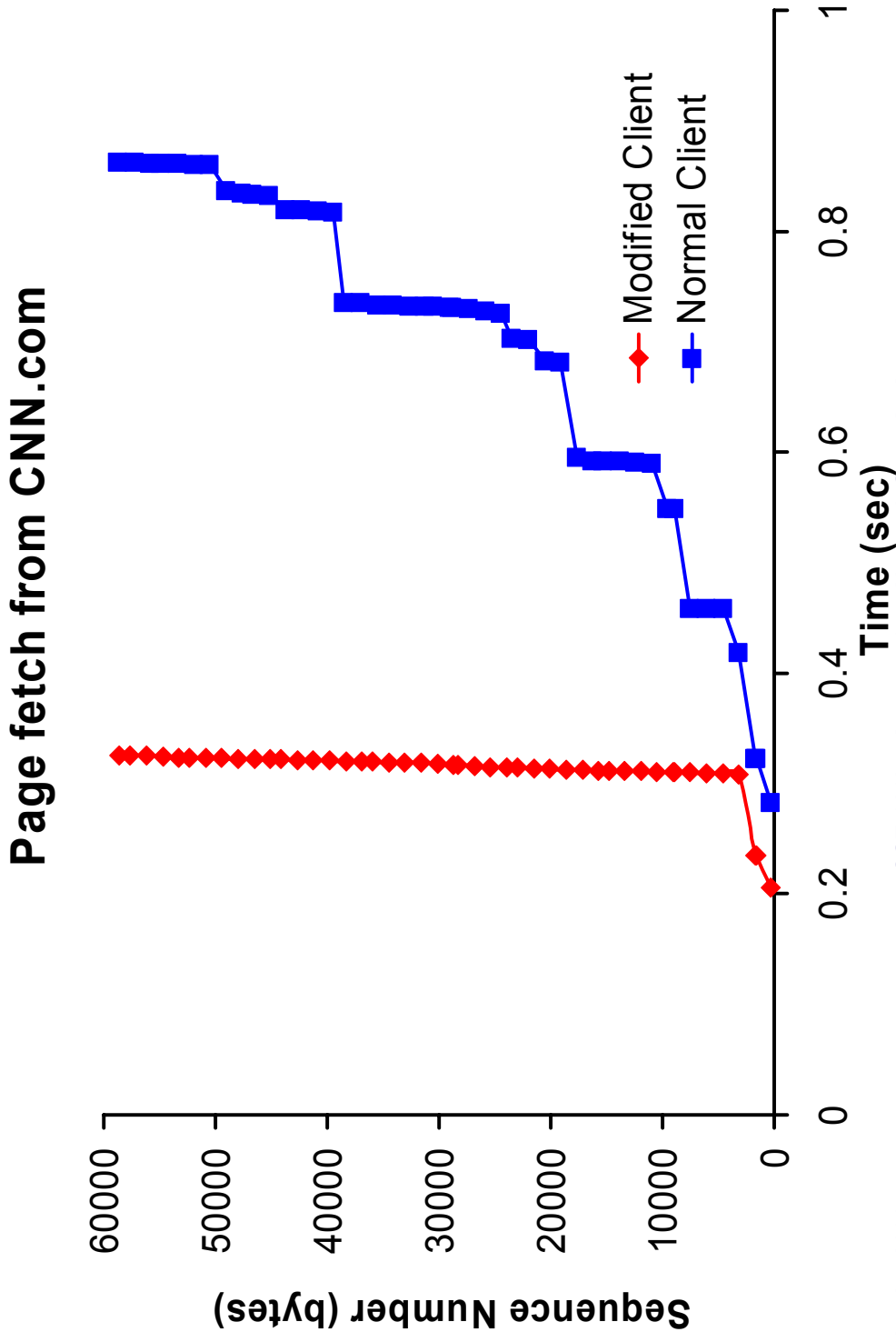
# (1) ACK Division

---



- Send M ACKs for one pkt
- Exponential growth factor proportional to M!
- Preserves end-to-end semantics

# Example



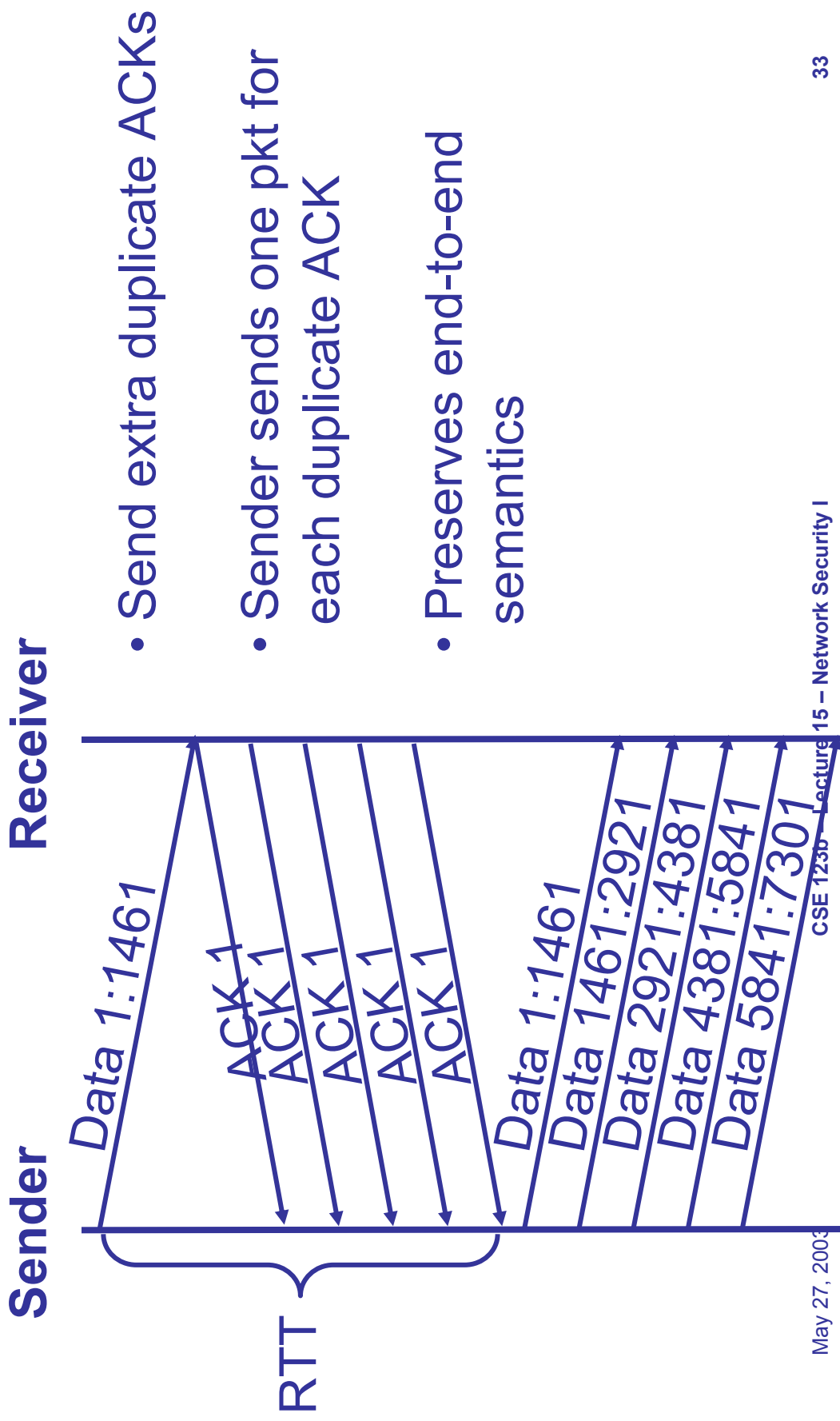
# Vulnerability 2: Fast Retransmit and Recovery

---

- Receive out-of-order segment => send duplicate ACK
- Sender receives 3 duplicate ACKs => fast retransmits, enters fast recovery
  - ♦  $Cwnd = cwnd/2 + 3 * SMSS$
  - ♦ On a duplicate ACK,  $cwnd += SMSS$
- Each additional duplicate ACK is taken as evidence that a data packet has left the network and therefore  $cwnd$  is increased

# (2) DupACK Spoofing

---



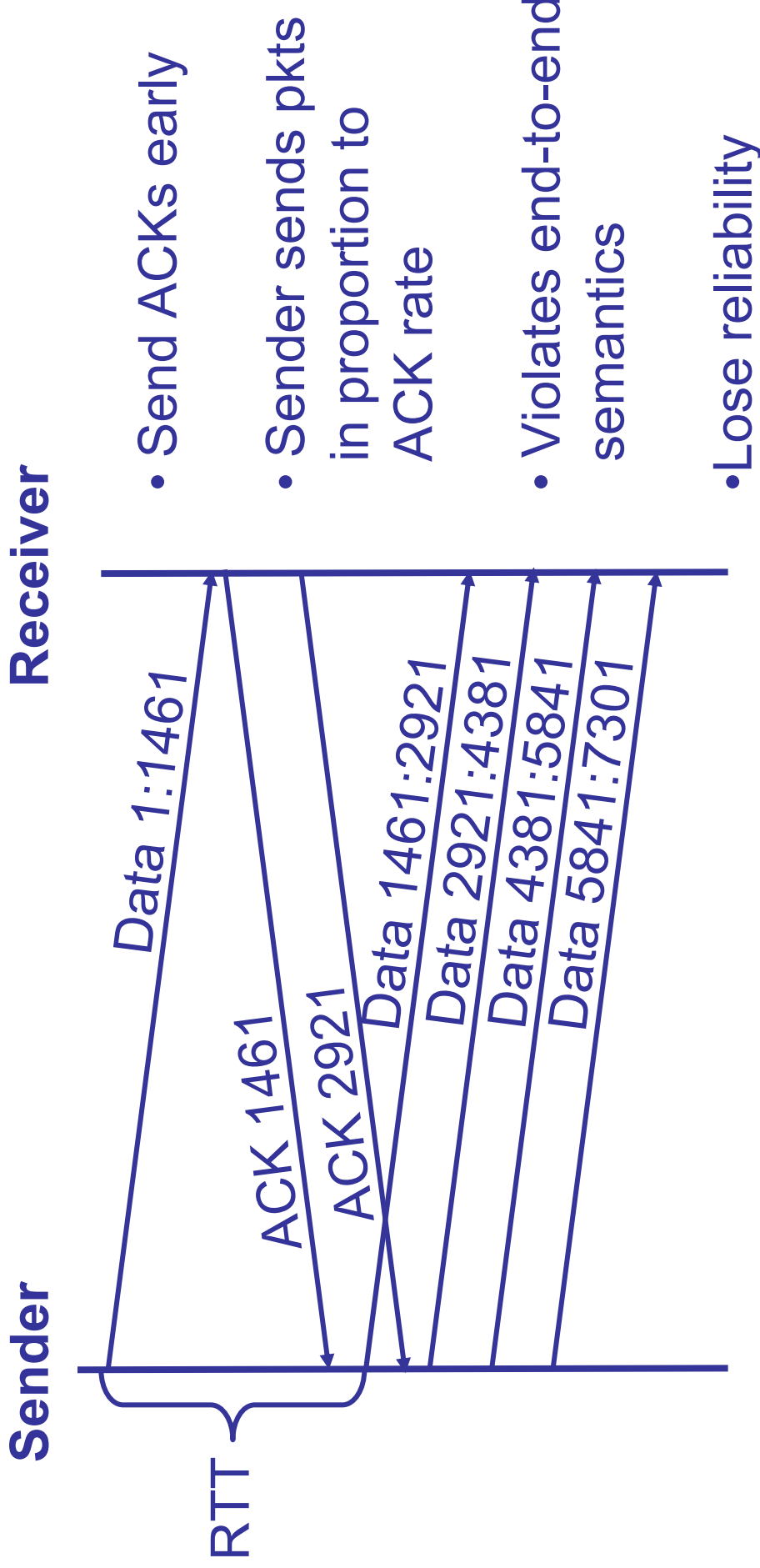
# Vulnerability 3: Freshness

---

- When sender receives a new ACK, it increases cwnd
- *But how do you know the receiver got the data?*
- Must recover at application layer
  - ♦ HTTP range request
  - ♦ FTP byte request

# (3) Optimistic ACKing

---



# Developing a solution

---

- Not well suited to cryptographic methods
  - ♦ Need to ensure **validity** of information, not authenticity or integrity
- Can't enforce behavior at remote peer
- **Solution:** penalize misbehavior
  - ♦ Artificially limit connection speed
  - ♦ Drop connection

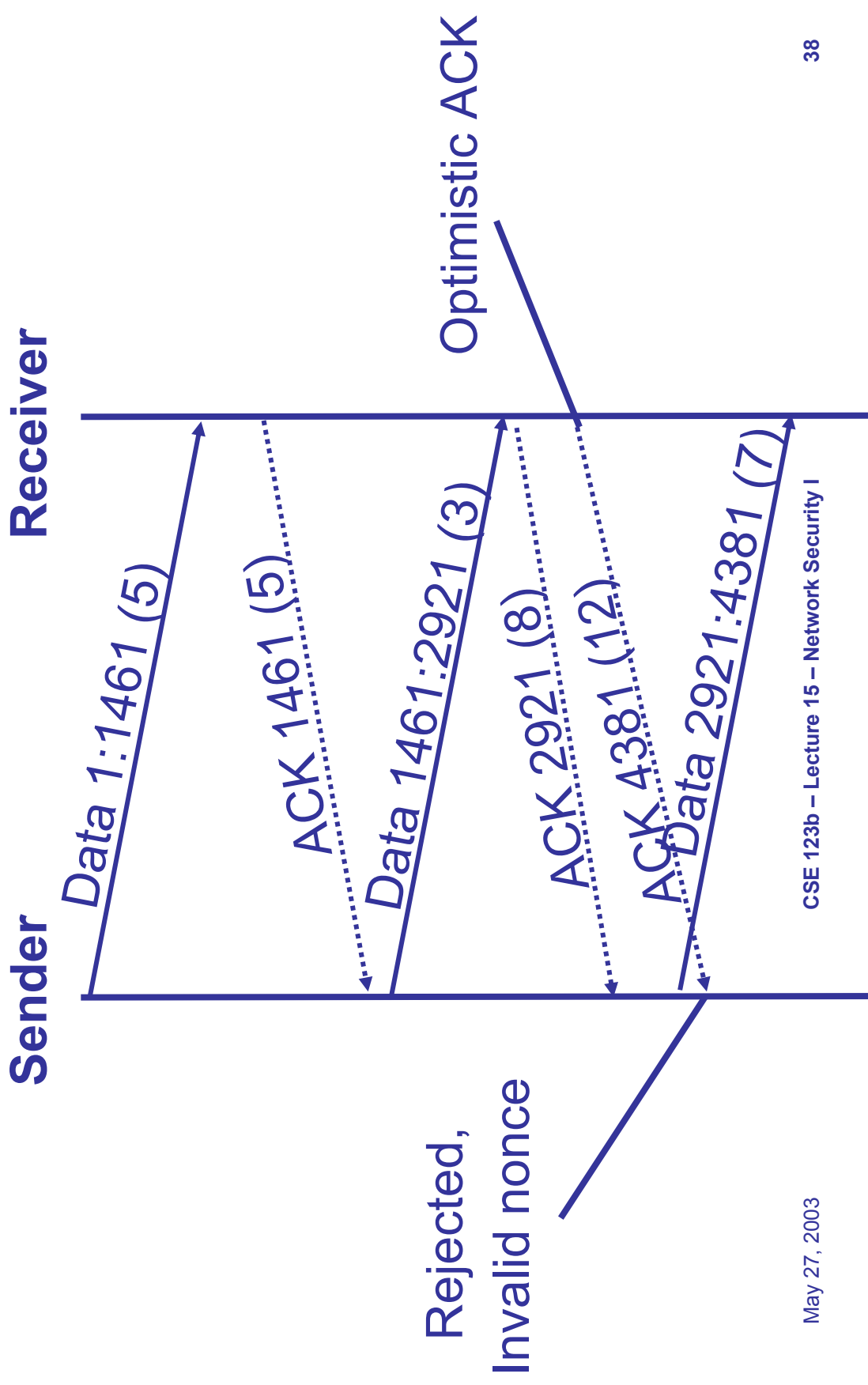
# Detecting misbehavior

---

- Eliminate sender assumptions
- Include extra “evidence” in ACK
  - ◆ Which data packet it was sent in response to
  - ◆ Proof of receipt and proof of freshness
- Mechanism: **cumulative nonce**
  - ◆ Sender puts a random # (nonce) in each pkt
  - ◆ Receiver echoes sum of nonces
  - ◆ Can be implemented probabalistically with a single bit

# Cumulative nonce example

---



# Summary

---

- Basic security
  - ◆ Cryptographic primitives and operations
  - ◆ Firewalls
  - ◆ Intrusion Detection
- Protocol vulnerabilities

# Next time

---

- More security...
- Denial-of-Service
- Worms