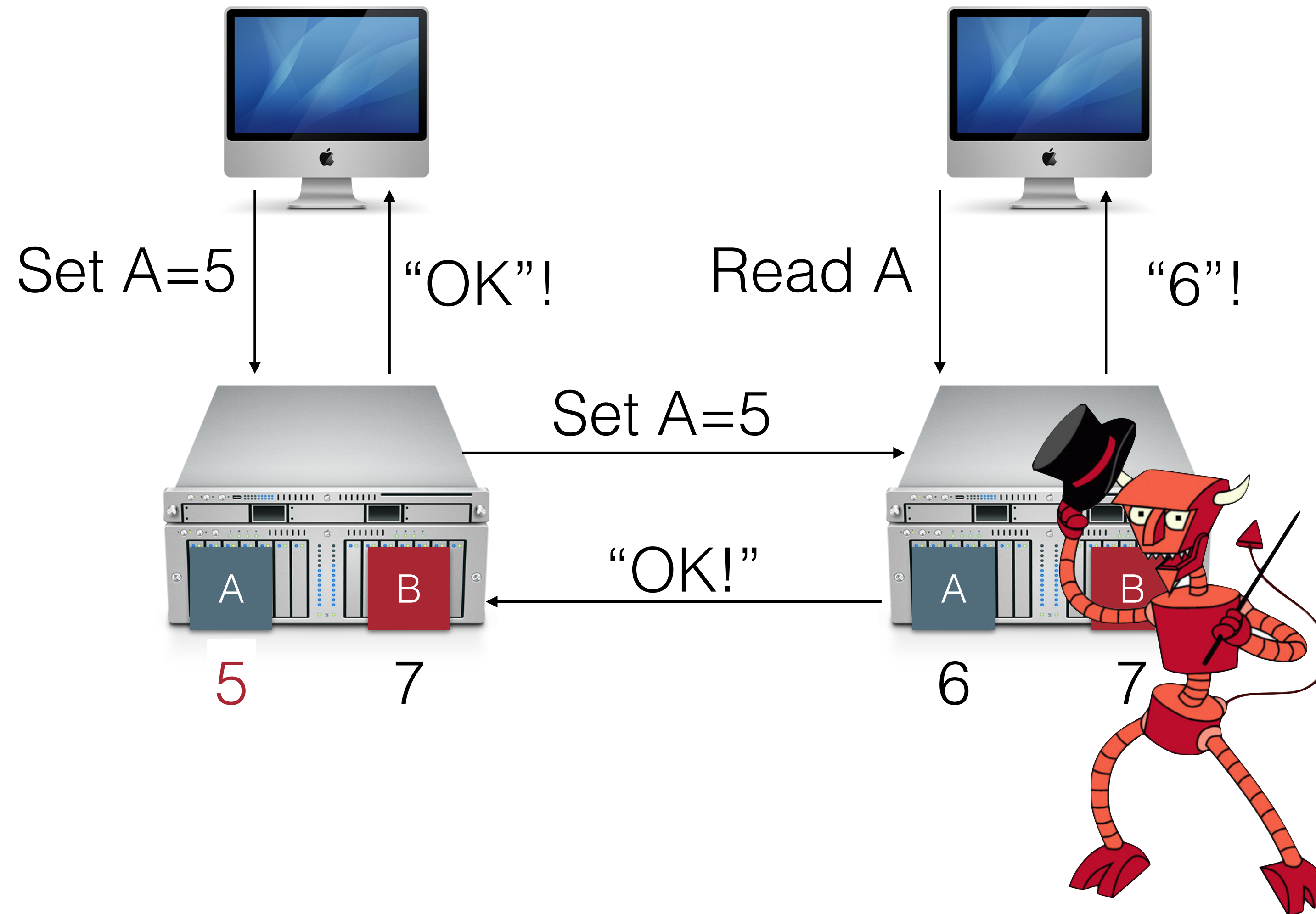


# Security in Distributed Systems

CS 475, Fall 2019

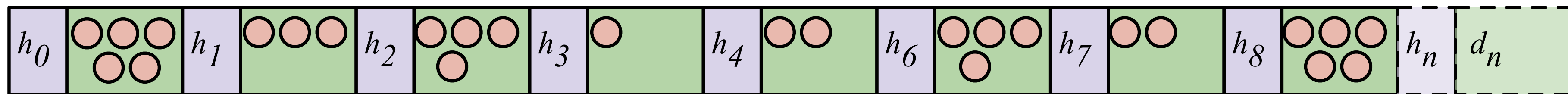
Concurrent & Distributed Systems

# Byzantine Faults



# Blockchains

- Solution: make it hard for participants to take over the network; provide rewards for participants so they will still participate
- Each participant stores the entire record of transactions as blocks
- Each block contains some number of transactions and the *hash* of the previous block
- All participants follow a set of rules to determine if a new block is valid



# Today

- Today:
  - Security in Distributed Systems
  - Discussion of course structure
- Final Exam info:
  - Primary focus is second half of semester (starting from Networks), still responsible for concepts from first half (notably mutual exclusion, locking granularity)
- Reminder - Project is out!
  - Fault-tolerant, sequentially consistent replicated key value store
  - Can do in a group (1 to 3 students per group)

# Security isn't (always) free

- You just moved to a new house, someone just moved out of it. What do you do to protect your belongings/property?
- Do you change the locks?
- Do you buy security cameras?
- Do you hire a security guard?
- Do you even bother locking the door?

# Security: Managing Risk

- Security architecture is a set of mechanisms and policies that we build into our system to mitigate risks from threats
- Threat: potential event that could compromise a security requirement
- Attack: realization of a threat
- Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise

# What does it mean for a distributed system to be secure?

- Maintain a secure channel between nodes:
  - Authenticity (Who am I talking to?)
  - Confidentiality (Is my data hidden?)
  - Integrity (Has my data been modified?)
  - Availability (Can I reach the destination?)
- Maintain some security about who participates in the system?
- What cryptographic tools are available to us?

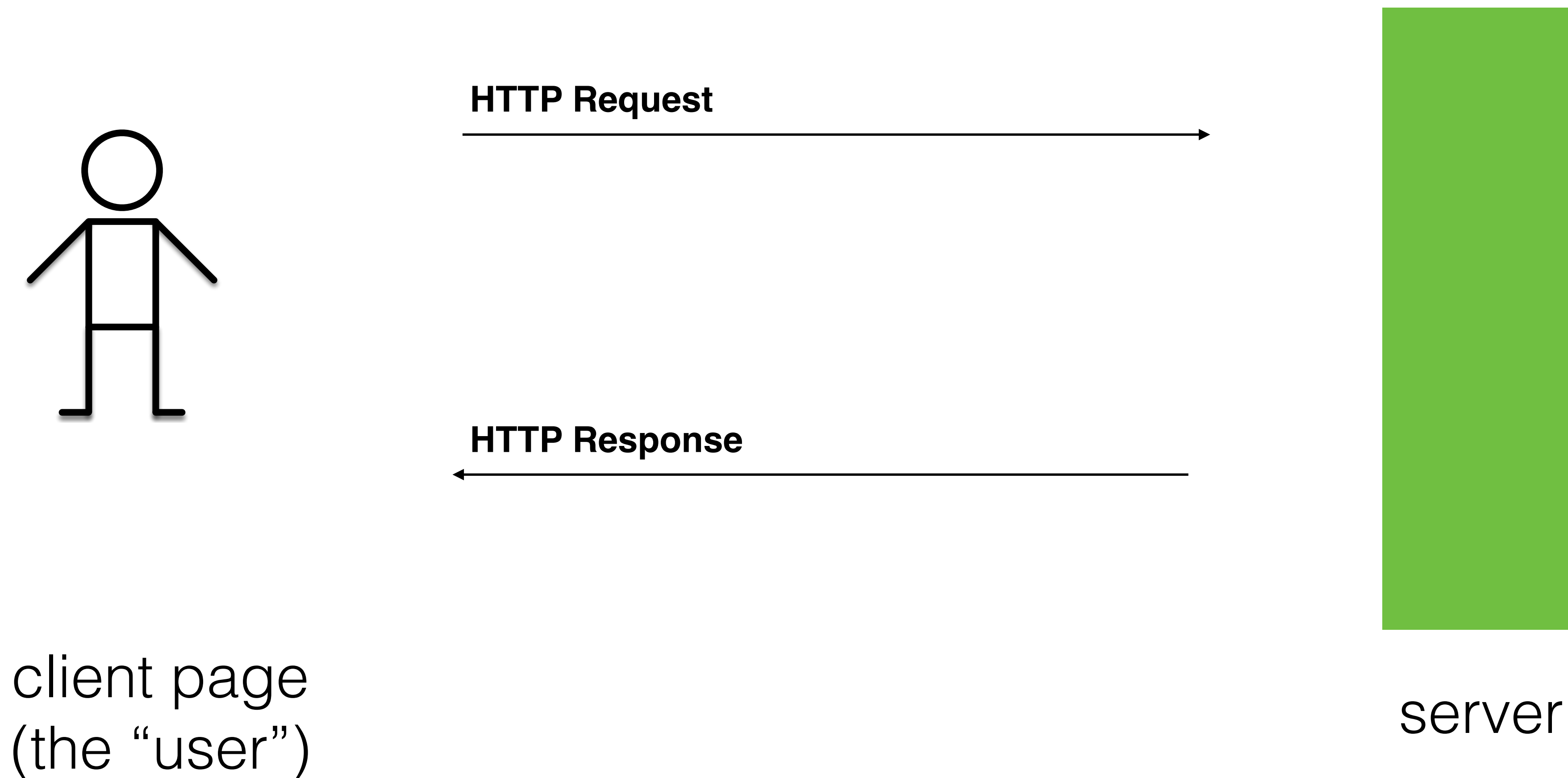
# Costs & Benefits

- Increasing security might:
  - Increase development & maintenance cost
  - Increase infrastructure requirements
  - Degrade performance
- But, if we are attacked, increasing security might also:
  - Decrease financial and intangible losses
- So: How likely do we think we are to be attacked in way **X**?

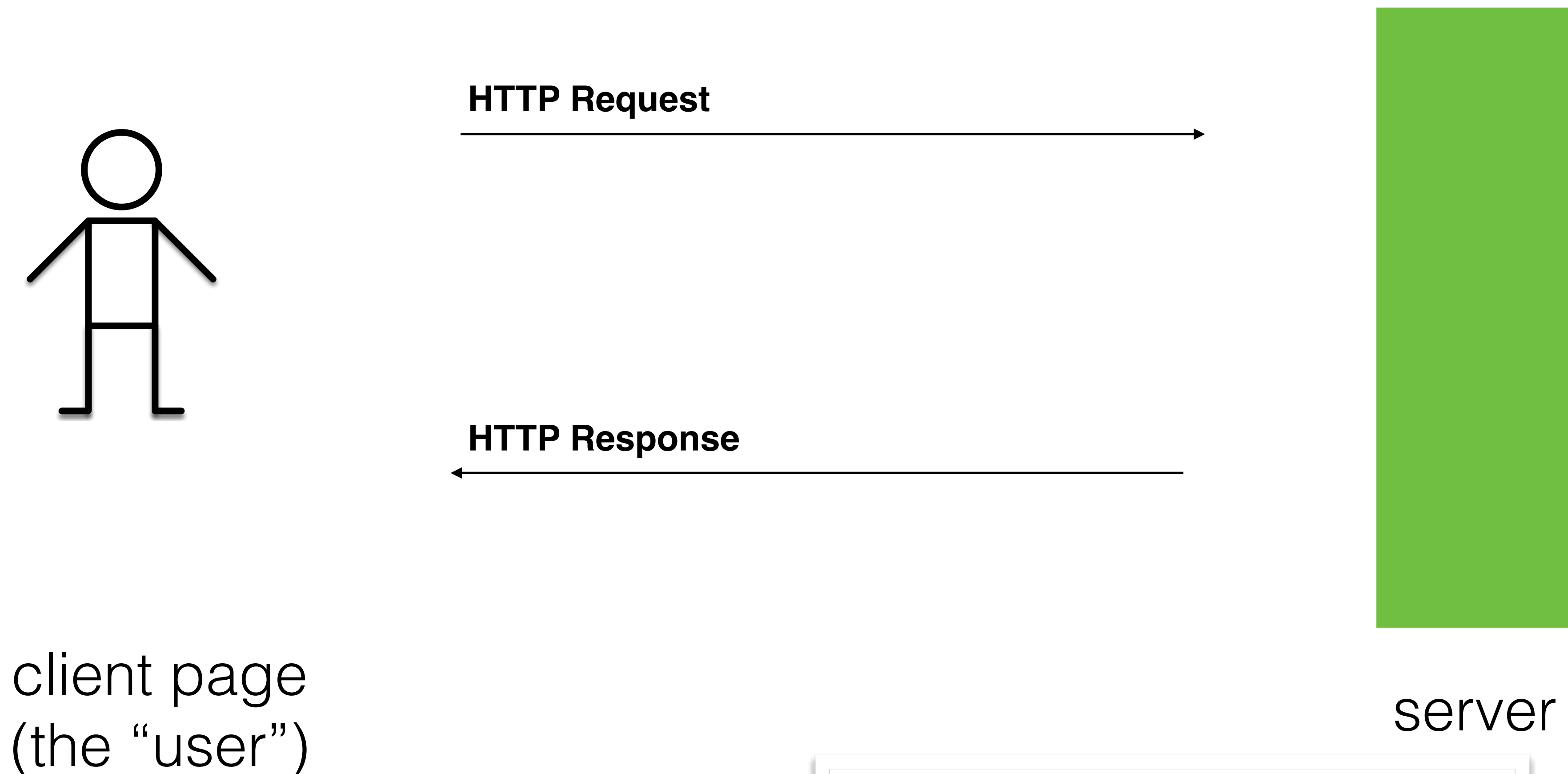
# Threat Models

- What is being defended?
  - What resources are important to defend?
  - What malicious actors exist and what attacks might they employ?
- Who do we trust?
  - What entities or parts of system can be considered secure and trusted
  - Have to trust **something!**

# Example Threat: Web Server

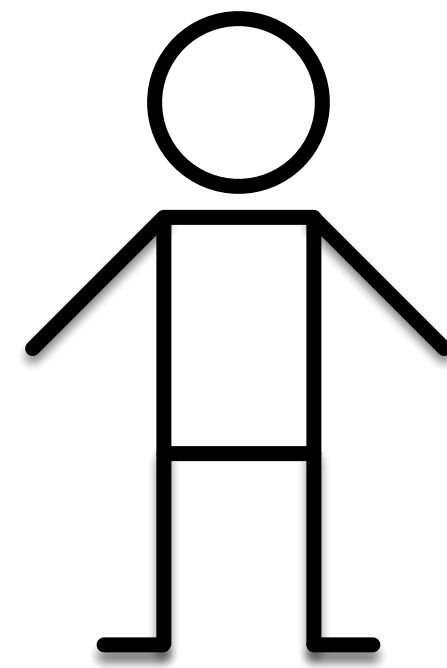


# Example Threat: Web Server



**Do I trust that this request *really* came from the user?**

# Example Threat: Web Server



client page  
(the “user”)

**Do I trust that this response  
*really* came from the server?**

HTTP Request



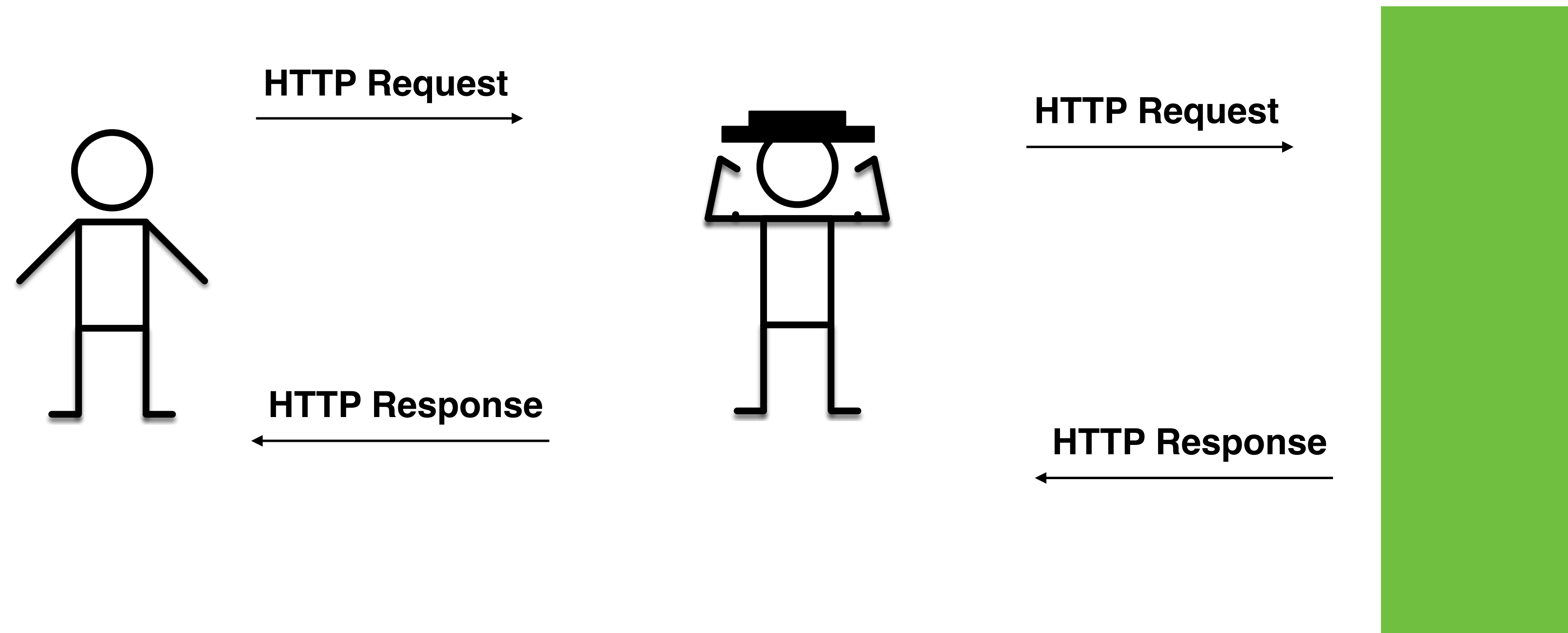
HTTP Response



server

**Do I trust that this request *really*  
came from the user?**

# Example Threat: Web Server



client page  
(the “user”)

malicious actor  
“black hat”

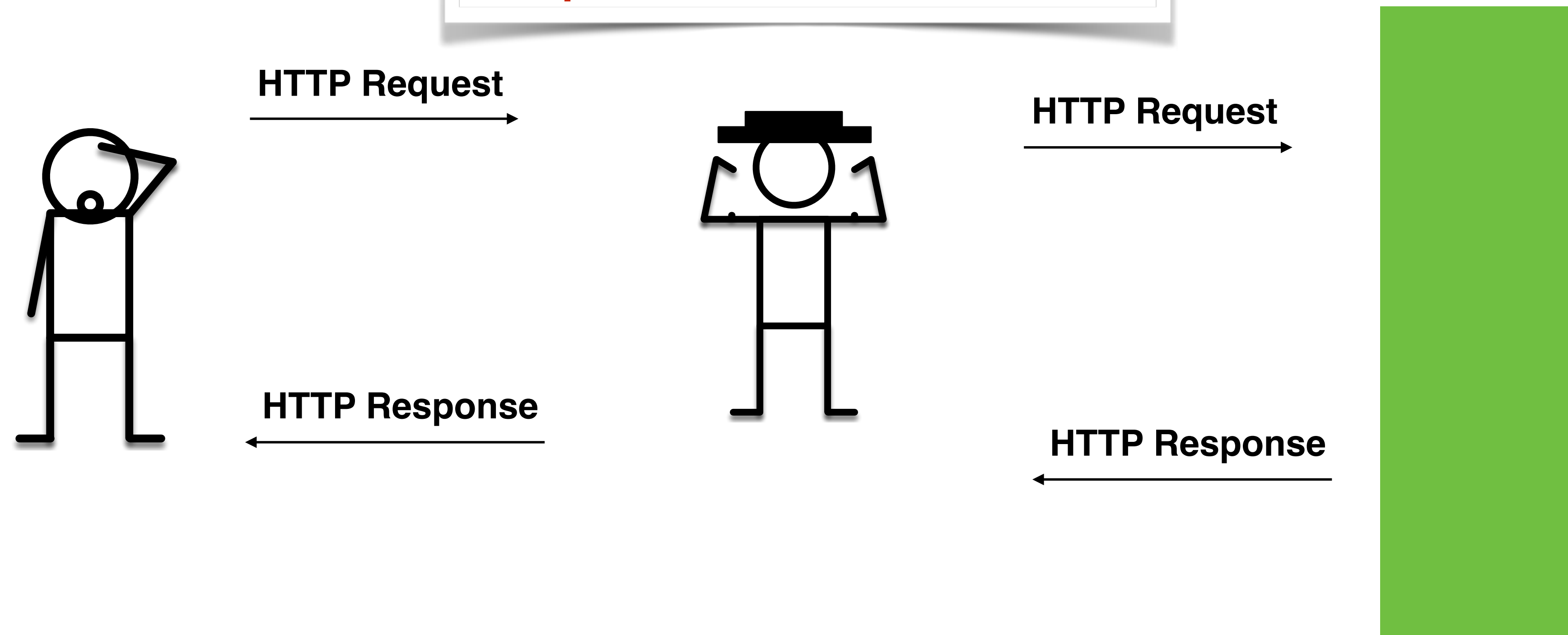
server

**Do I trust that this response  
*really* came from the server?**

**Do I trust that this request *really*  
came from the user?**

# Example Threat: Web Server

Might be “man in the middle” that intercepts requests and impersonates user or server.



client page  
(the “user”)

malicious actor  
“black hat”

server

Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

# Other Risks

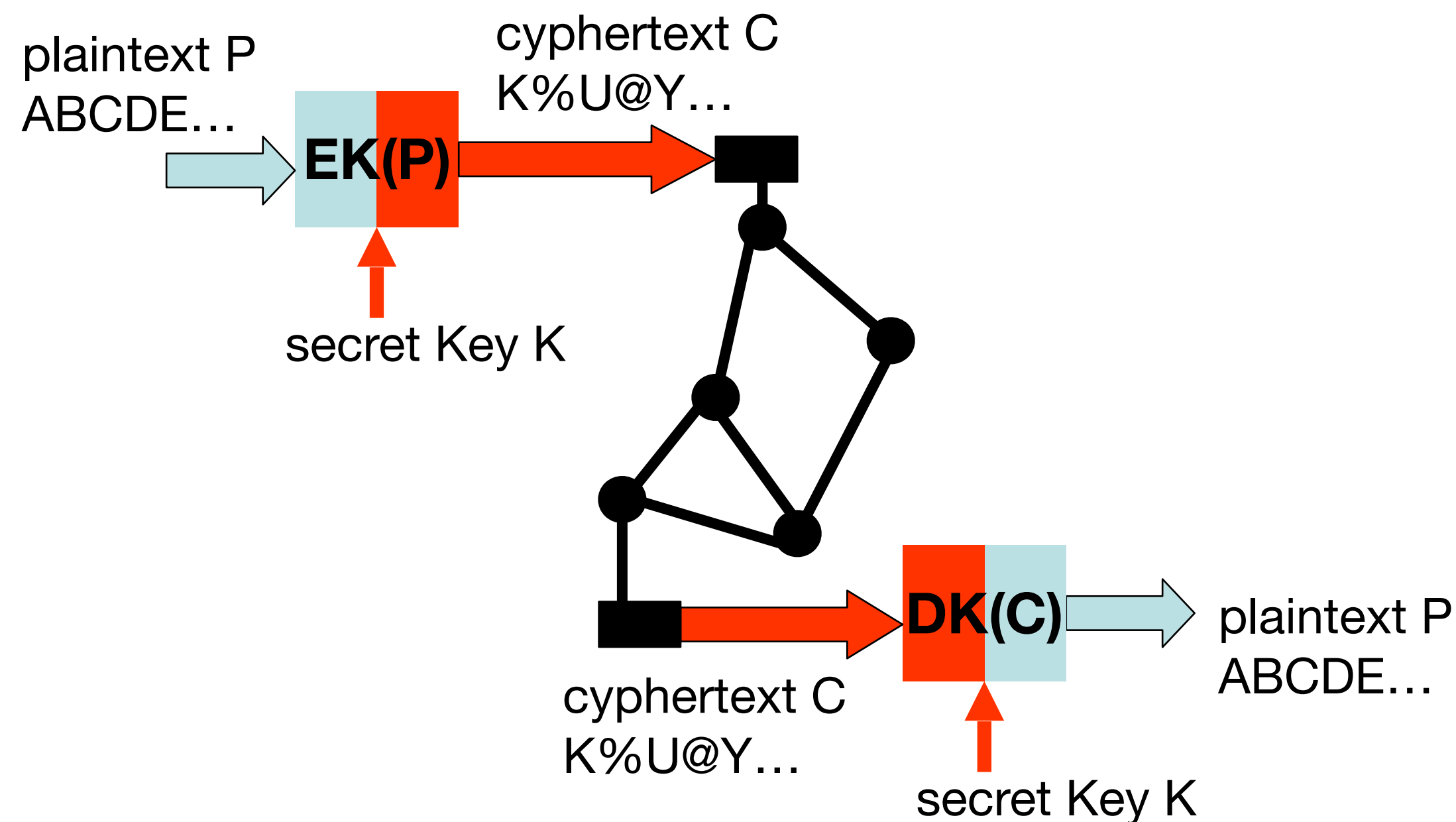
- Is our network well behaved?
- Is our network malicious?
- Who can access our system?
- Are our users well-behaved?
- Are our users malicious?
- Is our system well behaved?

# Protection Concerns

- Secure channels of communication
  - Authentication: is everyone who they say they are?
  - Confidentiality & integrity: is a third party interfering in our communication?
- Access Controls
  - Authorization: Who has access to an operation/resource?
  - Accountability: Maintaining an audit trail
  - Non-repudiation: A participant can not deny some action that they took with the system

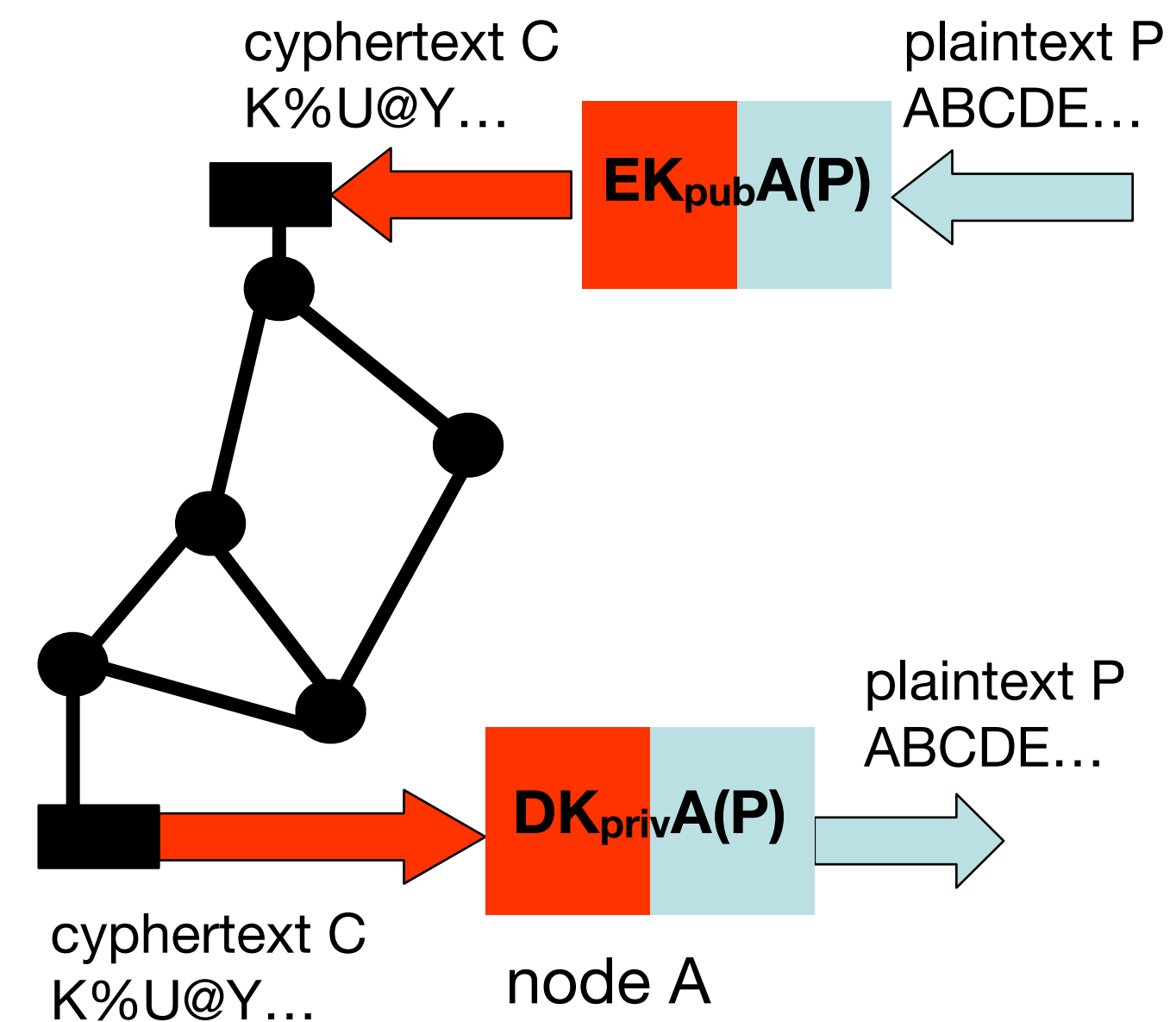
# Symmetric encryption, aka shared secret key

- $M = D_K(E_K(M))$
- $M$  is the data,  $D$  is decrypt,  $E$  is encrypt, and  $k$  is the key
- Computationally efficient (relatively)
- Can have hardware support too (e.g. iPhone)



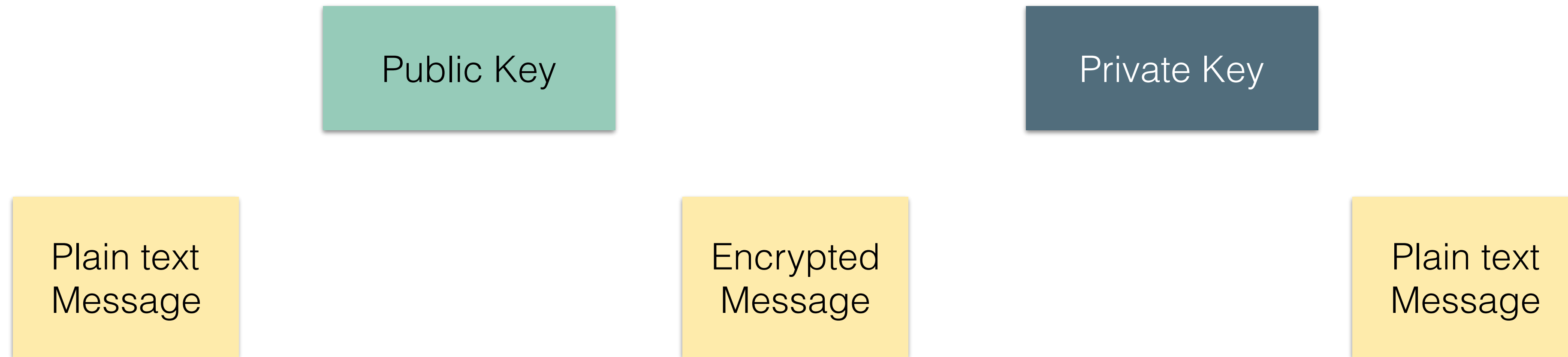
# Asymmetric encryption, aka public key/private key

- $M = DK_{priv} (EK_{pub} (M)) = DK_{pub} (EK_{priv} (M))$
- When a node B wants to send a message to node A, it obtains A's public key and uses  $K_{APublic}$  to encrypt the message
- Only A can decrypt the message using its private key
- Computationally expensive



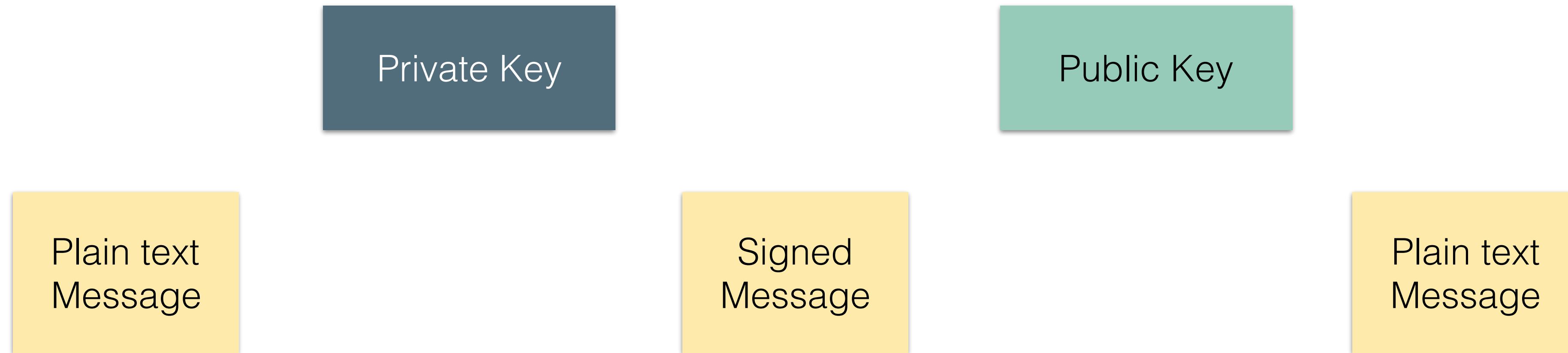
# Public/Private Key Encryption

- Encrypt with public key: only private key holder can decrypt



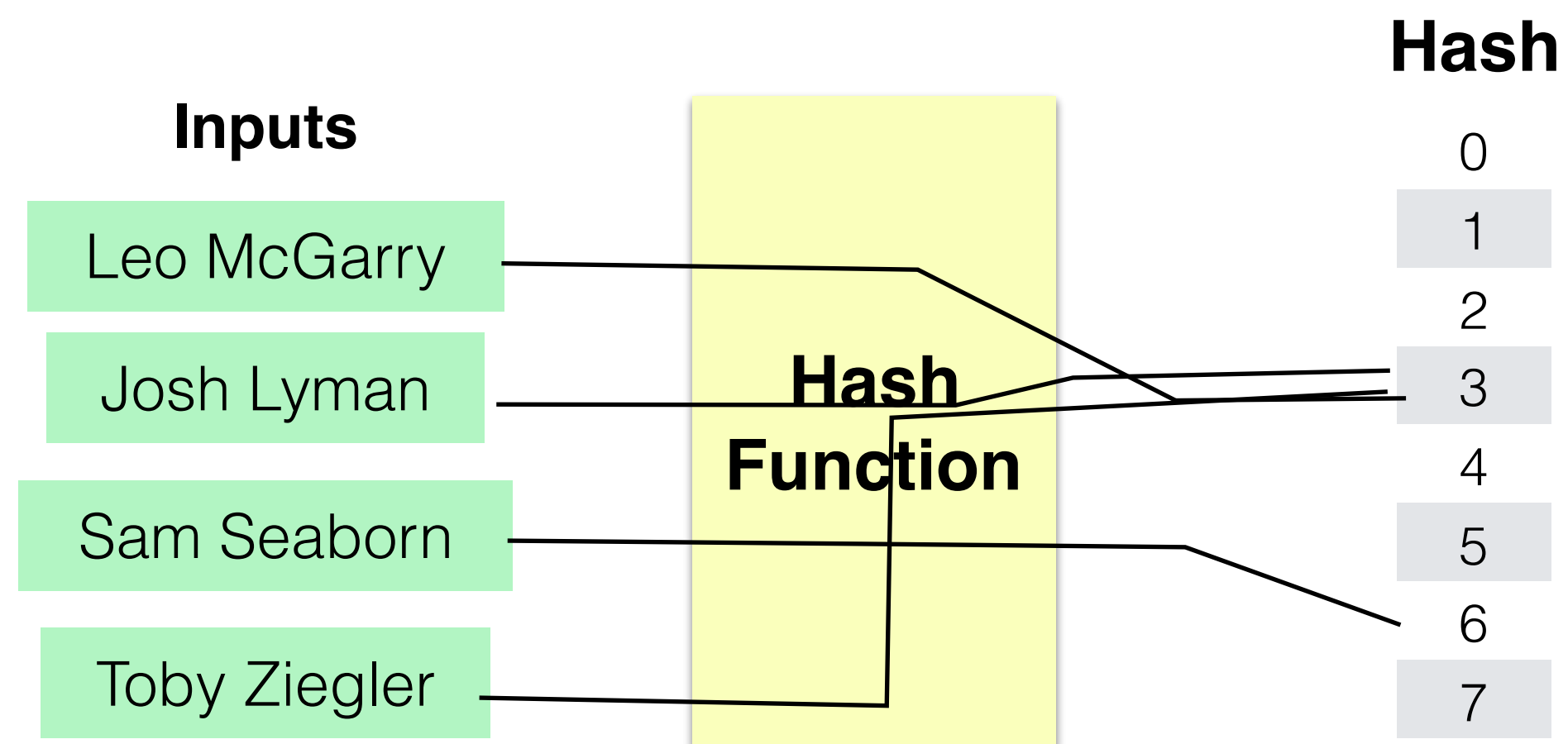
# Public/Private Key Encryption

- Encrypt with private key: anyone with public key can decrypt



# Hashing

- $S = H(M)$
- $S$ , aka digest, is a unique representation of data such that an accidental or intentional change to the data will change the representation
- Fixed size and independent of size of  $M$
- Computationally efficient



# Hashing to verify messages

- Just sending a hash of the message isn't enough!
- How do we know that a third party didn't tamper with the message *and* the hash?
- Solution: encrypt the hash using your private key. Anyone can verify that the hash was "signed" by you

# Symmetric vs Asymmetric Crypto

	Symmetric Crypto	Asymmetric Crypto
Requires a pre-shared secret	Yes	No
Relative speed	Very fast	Very slow

# Asymmetric Cryptography

- So, great: no need to pre-share anything, right!
- Widely used for instance... HTTPS! SSL!
- But: there's a bootstrapping problem
- When you visit amazon.com, the site will sign its content using its private key
- You can use amazon.com's public key to verify it's really from amazon.com
- How do you know what amazon.com's public key is though?
- “PKI” - Public Key Infrastructure

# Certificate Authorities

- A certificate authority (or CA) binds some public key to a real-world entity that we might be familiar with
- The CA is the clearinghouse that verifies that amazon.com is truly amazon.com
- CA creates a certificate that binds amazon.com's public key to the CA's public key (signing it using the CA's private key)

# Certificate Authorities

Amazon



amazon.com  
private key



amazon.com  
public key



Some world proof that we are  
really  
amazon.com

amazon.com certificate  
(AZ's public key + CA's sig)



amazon.com certificate  
(AZ's public key + CA's sig)

Certificate Authority



CA private key



CA public key

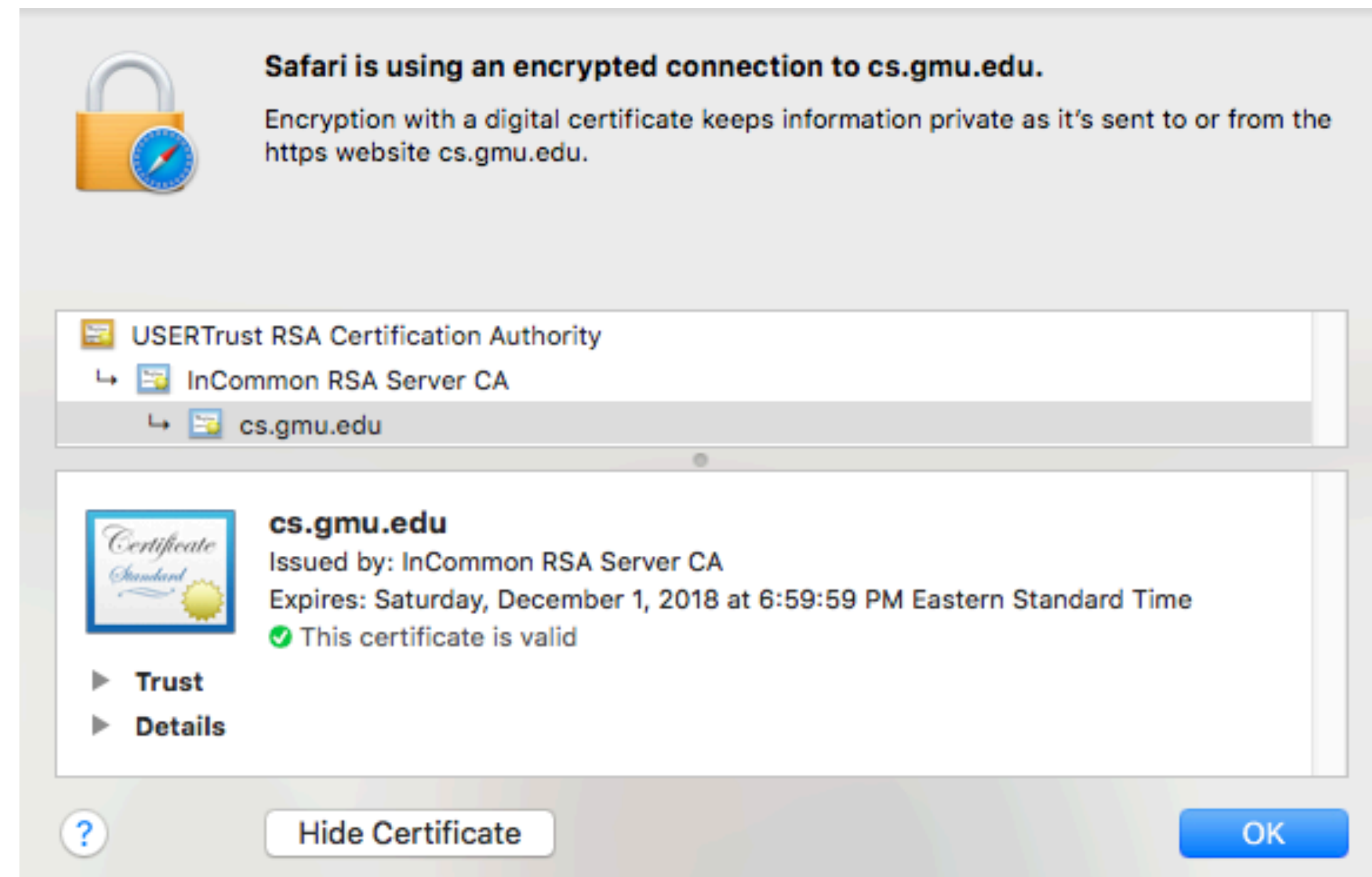
My Laptop



CA public key

# Certificate Authorities

- Note: We had to already know the CA's public key
- There are a small set of “root” CA's (think: root DNS servers)
- Every computer/browser is shipped with these root CA public keys



# Certificate Authorities


- What happens if a CA is compromised, and issues invalid certificates?
- Not good times.

## Security

### Fuming Google tears Symantec a new one over rogue SSL certs

We've got just the thing for you, Symantec ...

By Iain Thomson in San Francisco 29 Oct 2015 at 21:32

36  SHARE ▼

## Security

### Comodo-gate hacker brags about forged certificate exploit

Tiger-blooded Persian cracker boasts of mighty exploits



# Denial of Service Attacks

- A significant concern for distributed systems
- An attack on **availability** - attackers prevent legitimate users from accessing system
- Can attack:
  - Bandwidth
  - CPU
  - Memory
- Core problem:
  - Costs more to process a message than to send it

# Distributed Denial of Service Attacks (DDoS)

- Model: Attacker has (hundreds of?) thousands of machines at disposal to attack
- Most common form of DoS today
- Exhausts network bandwidth
- Typically rooted in a botnet - some command and control infrastructure setup by an attacker, who then controls all of these machines

# Strawman Defenses

- Make a filter list of bad addresses?
- Trace down the person responsible?

# Heuristic Defenses

- Overprovision
- Black-hole routing
- Filter anomalies
- Replication

# Overprovisioning

- Make a DDoS-proof site by making it far bigger than it needs to be
- Provision 100x bandwidth, 100x server capacity etc. compared to what you expect
- A losing battle: an attacker can always get more bots!

# Black-Hole Routing

- Limits the impact of an attack
- ISP re-routes traffic to the target site to a black hole
- Site still goes offline
- But not crashed, other sites on servers sharing network links are OK
- Most DDoS attacks are short-lived, so clears up later

# Anomaly Filtering

- DDoS traffic usually has something peculiar about it...
- Automatically generated requests following a pattern?
- Route all traffic through black-box filters that try to learn this stuff and identify anomalies
- Imperfect, but often works

# Other DoS attacks

- Reflector
- Complexity

# Reflector Attacks

- Exploits a publicly available service to amplify an attack
- Example: DNS
- Attacker makes a (relatively small) DNS request
- Attacker forges their own IP address with the victim's
- DNS server responds to the victim's IP address

# Complexity Attacks

- Increasingly common as we find defenses for other attacks
- Idea: Can I make one request that is 100 times as hard to process as other requests?
- Then I only need to make 1% of the requests I would have had to otherwise, in order to get the same attack!

# Billion lolz

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

**After parsing: this document contains “lol” repeated literally a billion times... ~3GB of RAM**

# Discussion of course structure

- First assignment without concurrency?
- More/less concurrency discussion at start?
- More/less programming?
- More/less detail on some topics?
  - Drop P2P?

# This work is licensed under a Creative Commons Attribution-ShareAlike license

- This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - for any purpose, even commercially.
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.