

# Fault Tolerance and Security

15-110 – Wednesday 10/30

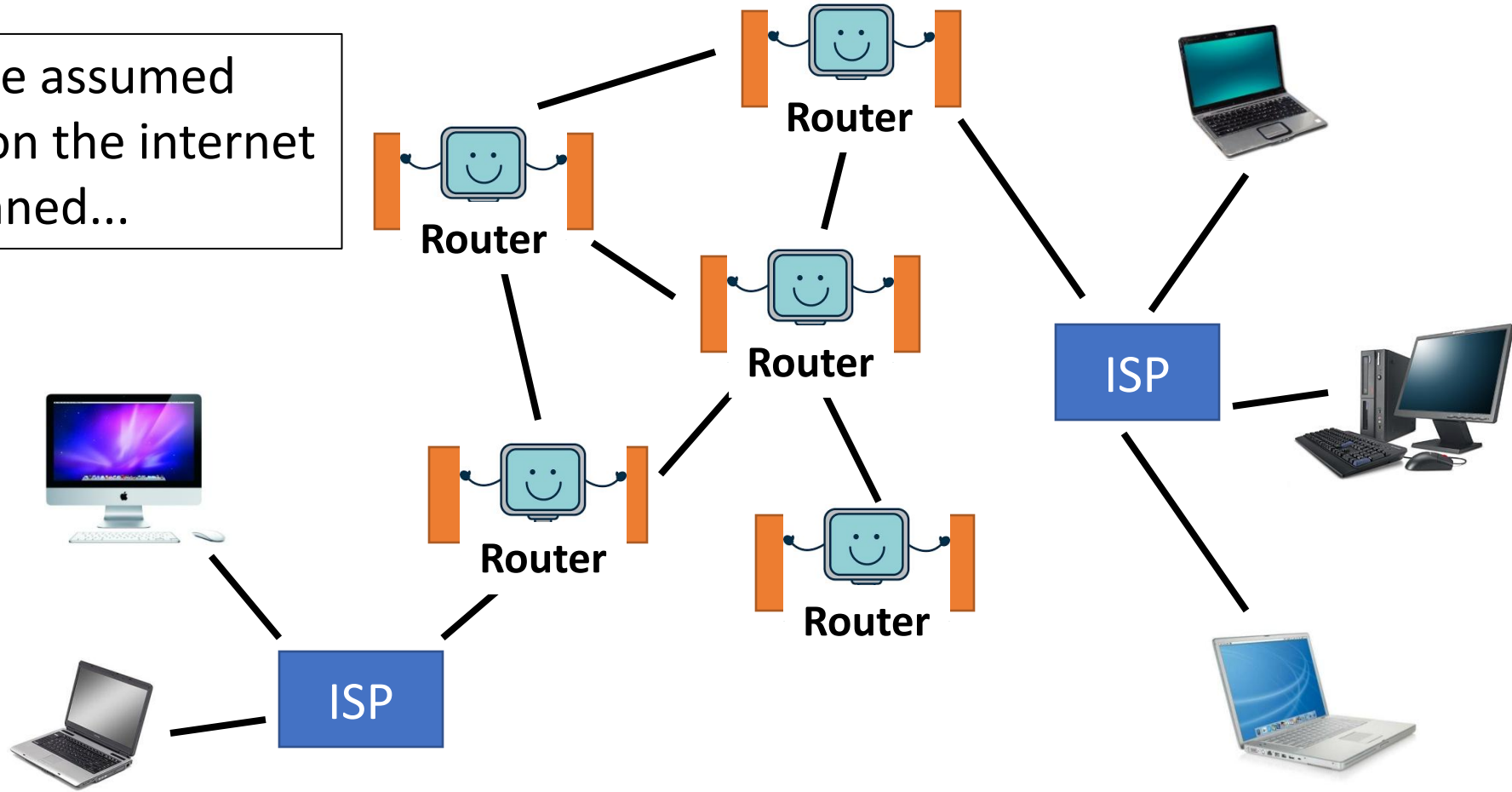
# Quizlet

# Learning Goals

- Recognize and define the following keywords: **fault tolerance, bottlenecks, net neutrality, data privacy, data security, DDOS attacks, and man-in-the-middle attacks**
- Recognize and define common approaches of **authentication**, including **passwords** and **certificates**
- Recognize and define the core elements of **encryption**, including **plaintext, ciphertext, keys, encoding, decoding, and breaking**
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA**, and recognize whether they are **symmetric** or **asymmetric**

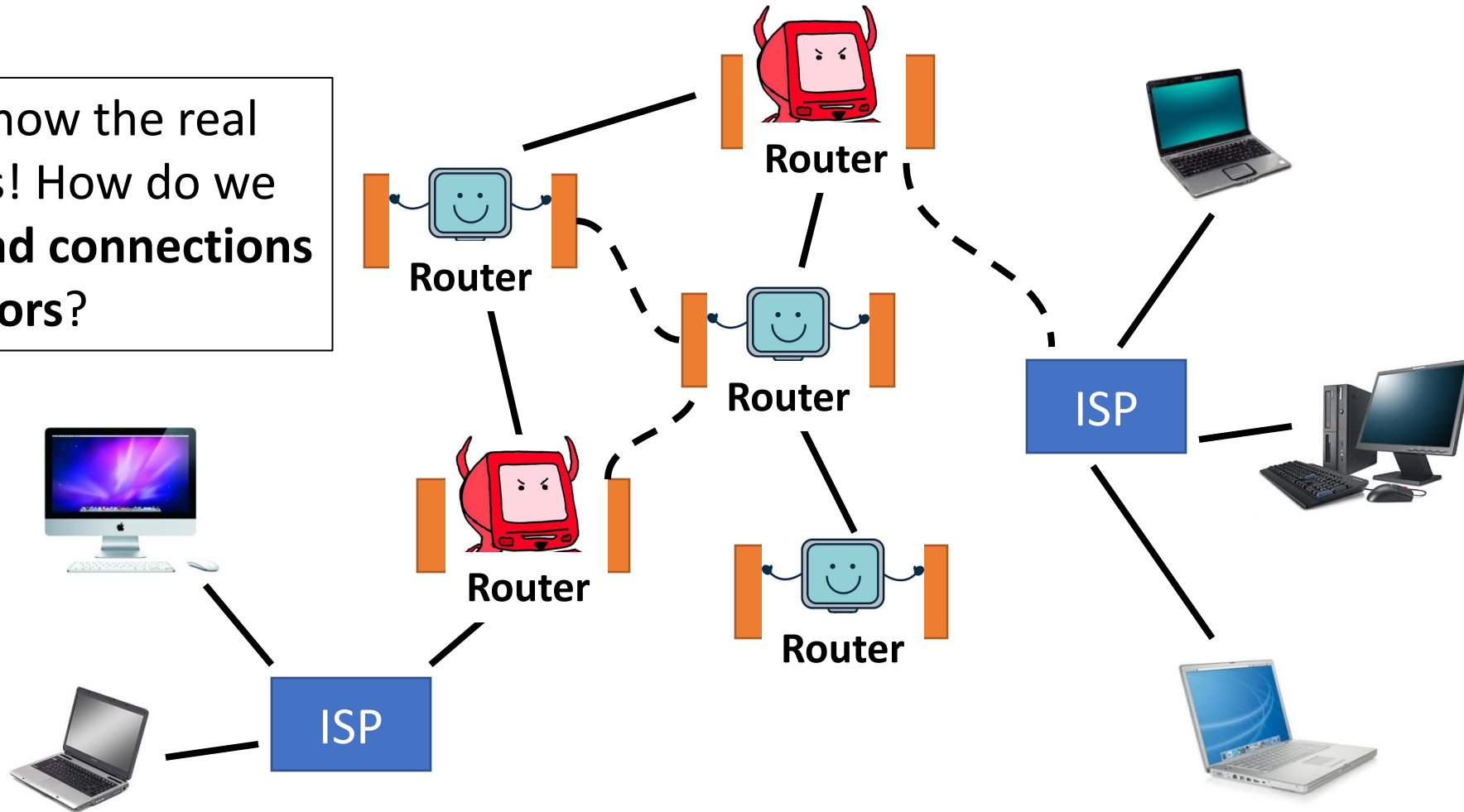
# The Internet: A Utopian Vision

Last time, we assumed everything on the internet goes as planned...



# The Internet: Reality

That's now how the real world works! How do we deal with **bad connections** and **bad actors**?



# **Fault Tolerance and Bottlenecks**

---

**Delivery of packets is unreliable** because the Internet is decentralized.

Can we rely on packets to show up properly? Not really...

- There are no guarantees that all packets will use the same route across the internet
- There are no guarantees that a computer will receive the packets in the intended order
- There are no guarantees that all the packets will arrive at your computer
- There are no guarantees that the packets will not be corrupted

To deal with all these potential problems, the Internet is designed to be incredibly **fault tolerant**.

This means that we **expect things to go wrong**, and there are plenty of backups and checks in place to fix things when that happens.

The way packets are delivered is fault tolerant; the way computers are networked to each other is fault tolerant; even the way data is stored in datacenters is fault tolerant! This is a core part of the design of any good distributed system.



Fault tolerance is important, because the more computers we have in a distributed system, the **higher probability somethings will crash.**

The probability that one computer randomly crashes while running a program is low (maybe 1 in 100,000).

But datacenters regularly run far more than 100,000 computers at the same time. **Crashes are expected and normal!**

To account for this, data is backed up on multiple machines; when one machine goes down, the data can be retrieved from another.



# Packet Fault Tolerance

**Q:** What happens if a **packet goes missing**? 

**A:** Your computer knows how to put packets back together based on the data they carry. Most protocols can tell if a packet is missing. If it is, the browser simply sends another request for a new set of packets.

**Q:** What happens if a **packet is corrupted**? 

**A:** Every packet contains a **checksum** that the computer can check to make sure it's not corrupted. If it is corrupted, the computer just sends a request for a new set of packets.

# Network Fault Tolerance

**Q:** What happens if **your computer goes down**? 

**A:** This is bad for you, but it's fixable and happens all the time! When your computer restarts, it can pick up where it left off, sending new requests for the data it didn't get. Usually you can return online quickly.

**Q:** What happens if a **company's website (a server) goes down**? 

**A:** Most companies have many servers that can all handle traffic to the same website, so traffic to the server that is down gets re-routed.

If **all** of a company's servers go down, then the website goes down too.

# Network Fault Tolerance

**Q:** What happens if a **router goes down**? 

**A:** This is fine – traffic will just be sent to other routers instead. The core of the internet is **heavily connected** and **decentralized**, so this will not disturb traffic.

**Q:** What happens if a **DNS Server goes down**? 

**A:** There are lots of DNS Servers spread across the world. If one goes down, your request gets sent to a different one.

# Network Fault Tolerance

**Q:** What happens if your ISP goes down? 

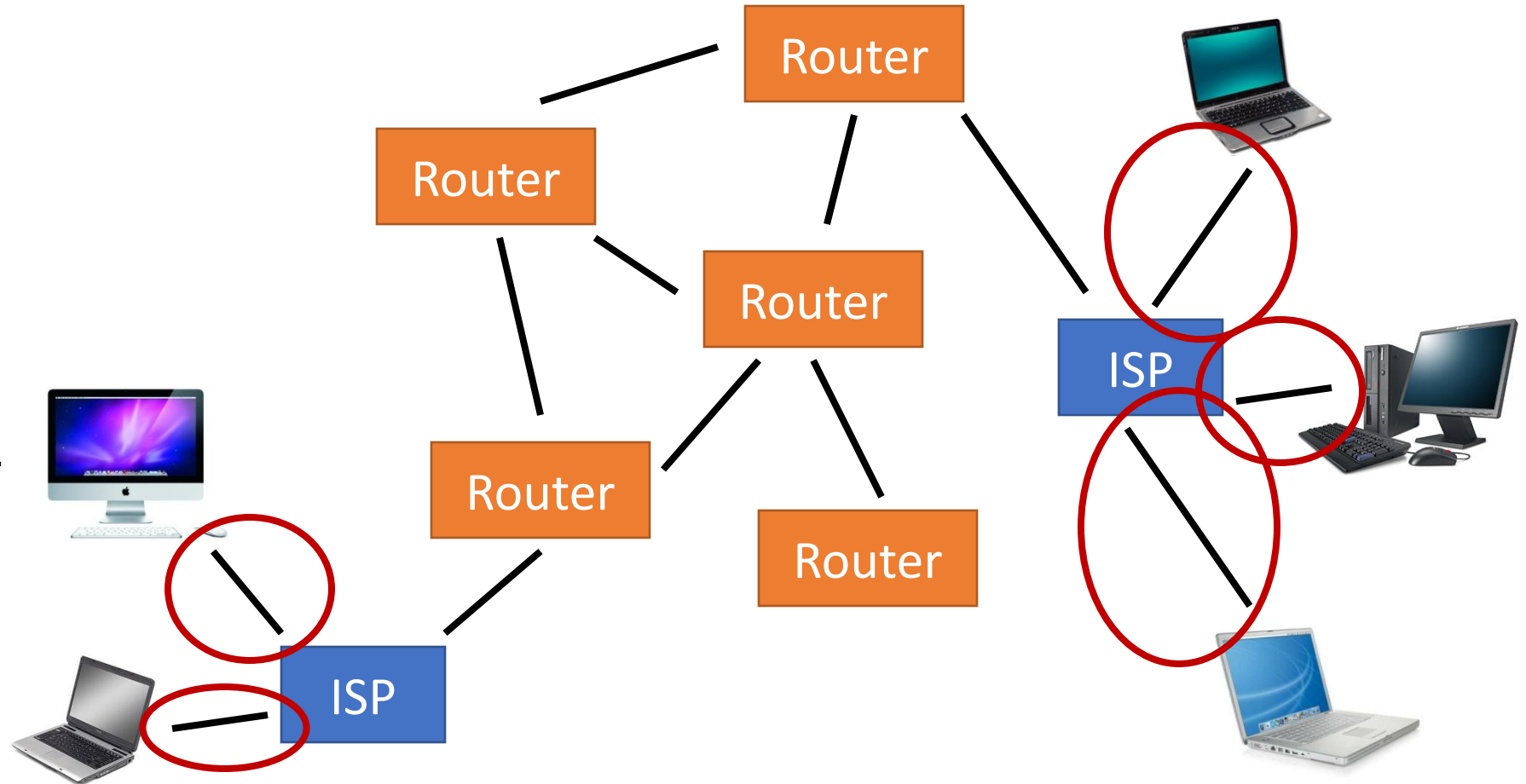
The user must distribute all their packets to the ISP first, who sends them on to the rest of the Internet. All response packets must pass through the ISP to reach the user.

[if time] **Discuss:** how does this affect the way the Internet functions?

# There is no one point of control for the Internet.

It's hard for one organization to control the whole internet, because the core of the internet contains no single point of control for everyone.

That's not true for your local connection – your ISP is a **bottleneck** to the rest of the internet. (So is your local computer, but you usually have control over that).



**Governments can shut off the Internet for an entire country** when they control the ISPs.

If all the ISPs shut down traffic, local computers have no way to access the broader internet. It's still there – it's just not connected.

If there is only one main connection between the core of the Internet and a country (like a single router that serves as the general entry point), the government can also shut down the internet if they shut down that router.



Government shutdowns are **becoming increasingly common**. Check out [this video](#) from 2020 for more information about why and how that works.





**Net Neutrality** is a principle which states that ISPs must treat all internet traffic **equally**.

ISP control over your local internet connection also relates to **net neutrality**, a term you may have heard used in various political debates.

Packets should not be prioritized or de-prioritized based on who sent them, who is receiving them, or what is in them.

In terms of policy, Net Neutrality states that Internet access should be considered a utility, like phone line connections.

Without Net Neutrality, an ISP could ask a website that sends a lot of packets (like YouTube) to pay them for the extra work.

Without Net Neutrality, an ISP could ask a website that sends a lot of packets (like YouTube) to pay them for the extra work. If the company refused, the ISP could **throttle** the track to that website's traffic to make it appear slower to the user. On the other hand, if the company pays, maybe they get prioritized instead.

The ISP could also offer deals to their customers based on the websites they visit. For example, Verizon might make your monthly bill cheaper if you only visit websites on a Verizon-approved list. In an extreme example, an ISP could entirely block your access to a website it doesn't approve of.

Net Neutrality is currently not law in the United States (except for in some states like [California](#)). It is law in some other countries, like India.

# **Data Privacy and Security**

---

Data privacy and data security have a common goal: **no third party should be able to read certain types of data** being sent across the internet.

**Data privacy** is the idea that we might want to have control over our data, both who has access to it and what others do with it. Privacy is important to people for personal, cultural, and safety reasons.

**Data security** is the idea that we want to keep some communications secure and reliable; in other words, no one other than the sender and the receiver should be able to read or modify the data. Security is important across a range of interactions, like financial transactions or confidential briefings.

**Adversaries** are malicious actors (people, organizations, governments) who want to prevent users from achieving their goal.

One example of malicious behavior is trying to **collect data that users want to keep private or secure.**

Adversaries may have varying **objectives, resources, and experience**, but all of them want to get access to data that the data-holder doesn't want them to have.

The Internet has three characteristics that make attacks common and give adversaries protection.

**Automation:** you can write a program to repeat an action indefinitely

**Action at a distance:** you do not need to be physically present to start a security attack

**Technique propagation:** it is easy to distribute and share malicious code to other adversaries

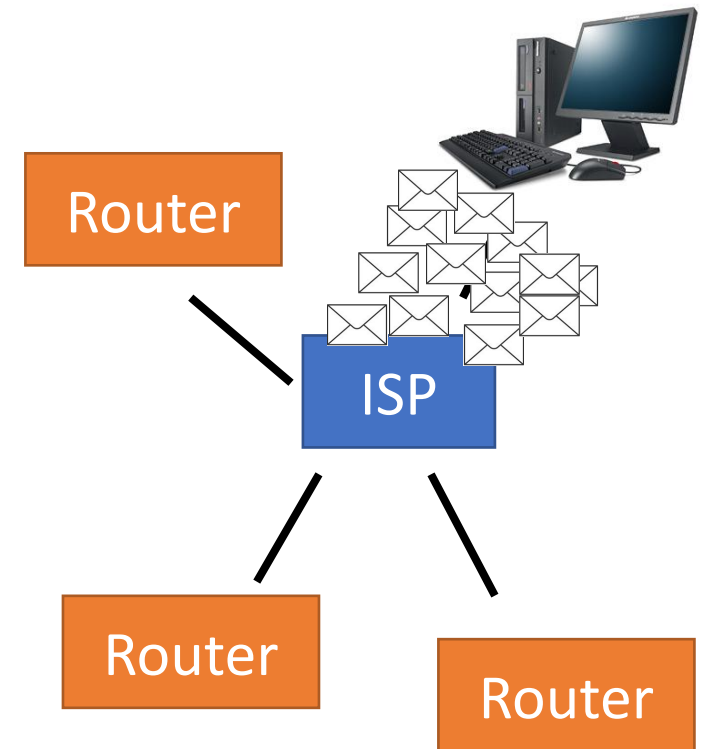
In a **Distributed Denial of Service (DDOS) attack** an adversary sends a huge amount of data to a server to overwhelm it.

In a DDOS attack, **adversaries send a huge amount of data** to a server within a short period of time (as a large number of packets).

This overwhelms the server and **makes it impossible for it to respond to authentic requests**, so the site looks like it is down to a normal person.

DDOSing can also happen accidentally when a lot of people suddenly start sending requests to a single site.

Analog: This is like a flash mob trying to get into your favorite small business – none of the regular customers can get through the door.



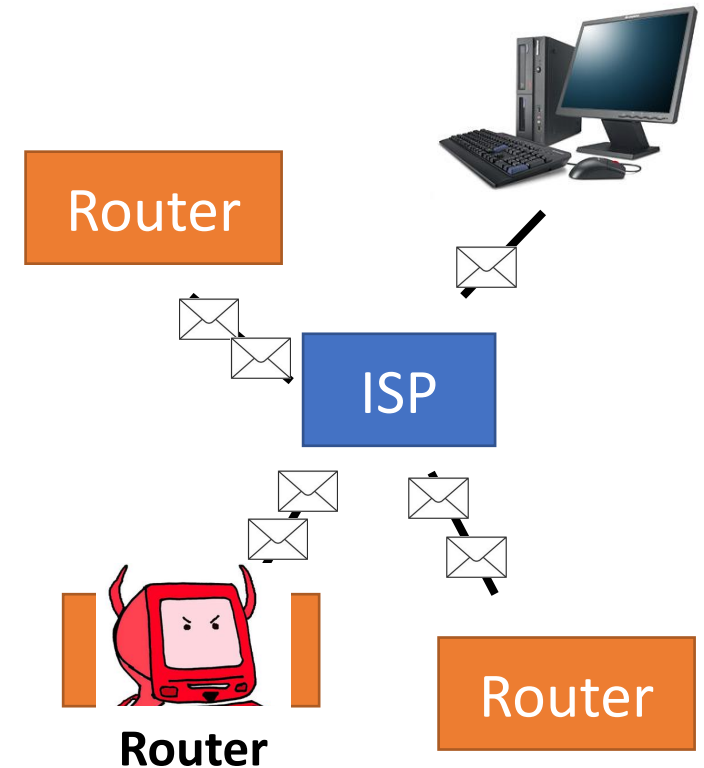
In a **Man-In-The-Middle attack** is when an adversary inserts themselves between two users.

An adversary sets up a router in a network that pretends to be a normal router. That lets this evil router intercept packets that are being sent to different destinations.

An adversary can read the data being sent by others and even change it, since the packets use standard protocols.

**Encryption** can help prevent these attacks.

Analog: This is like a postal worker reading and perhaps changing the message you write on a postcard.





We can use authentication and encryption to **protect data from adversaries**.

We have algorithmic ways to protect our data from adversaries:

**Authentication** to verify users are who they say they are.

**Encryption** keep data secret.

# Authentication

---

**Authentication** is a process that confirms someone is who they say they are.

It is used in any situation where you want to verify someone's identity on the internet.

You authenticate your identity every time you log into an account online.

Websites also authenticate their identity during secure online communications.

We often do authentication with usernames and **passwords**.

When you log in to a service online, you provide a username and a password. **The username is the identity you're claiming; the password is the verification.**


In addition to text passwords, we can use biometric data or physical tokens.

Carnegie Mellon University

Web Login

AndrewID

Password



Warning: The URL for this page should begin with **https://login.cmu.edu**.  
If it does not, do not fill in any information, and report this site to [it-help@cmu.edu](mailto:it-help@cmu.edu).

[About](#) | [Change Password](#) | [Forgot Password?](#)

An adversary could try and figure out the password using a **brute-force approach**.

Adversaries can try all possible passwords, but we know that that isn't efficient. If the password allows for lowercase, uppercase, number, and symbol characters, an  $n$ -character password takes  $94^n$  guesses to crack.

For a password that has 10 characters, there are  $94^{10}$  possible combinations of characters. That is larger than the number of grains of sand on Earth. That's insanely inefficient!

In reality, adversaries can figure out passwords using **human and security vulnerabilities**.

### **Human vulnerabilities:**

- Dictionary attacks, where adversaries guess dictionary words (as users often include real words in passwords)
- Social engineering, where the adversary tries to trick someone into revealing their password.

### **Security vulnerabilities:**

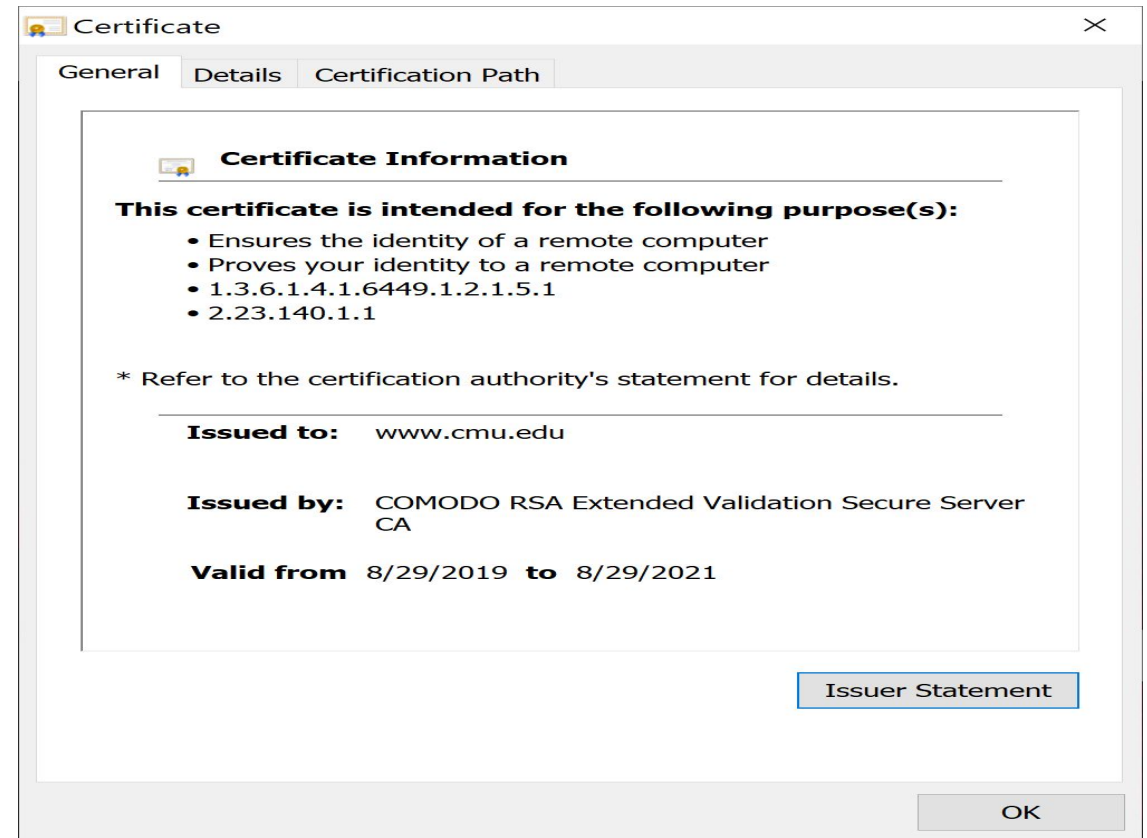
- Password databases are often 'leaked' this way. Many people reuse passwords across multiple accounts, so adversaries will try to match their leaked passwords to other accounts

If you have a password that is not a dictionary word, is reasonably long, and isn't used on other websites, it is very hard for someone to "hack" your account. You can also use a password manager!

Websites authenticate their identity using **certificates**.

A certificate is data that confirms the server you're communicating with actually owns the website it's claiming to be. You can view a website's certificate by clicking on the lock icon to the left of the URL in your browser.

Your browser establishes a secure connection to a website by first checking that it has a valid certificate.



Certificates are issued to websites by **Certificate Authorities.**

These organizations verify the identity of the website by communicating with people in the real world, then issue the website a certificate.

Certificates usually have an expiration date. When the certificate expires, the organization needs to verify itself again.

Browsers keep lists of trusted certificate authorities.



# Encryption

---

**Encryption** is the process of **encoding data** so that only the sender and the receiver can read the data's message.

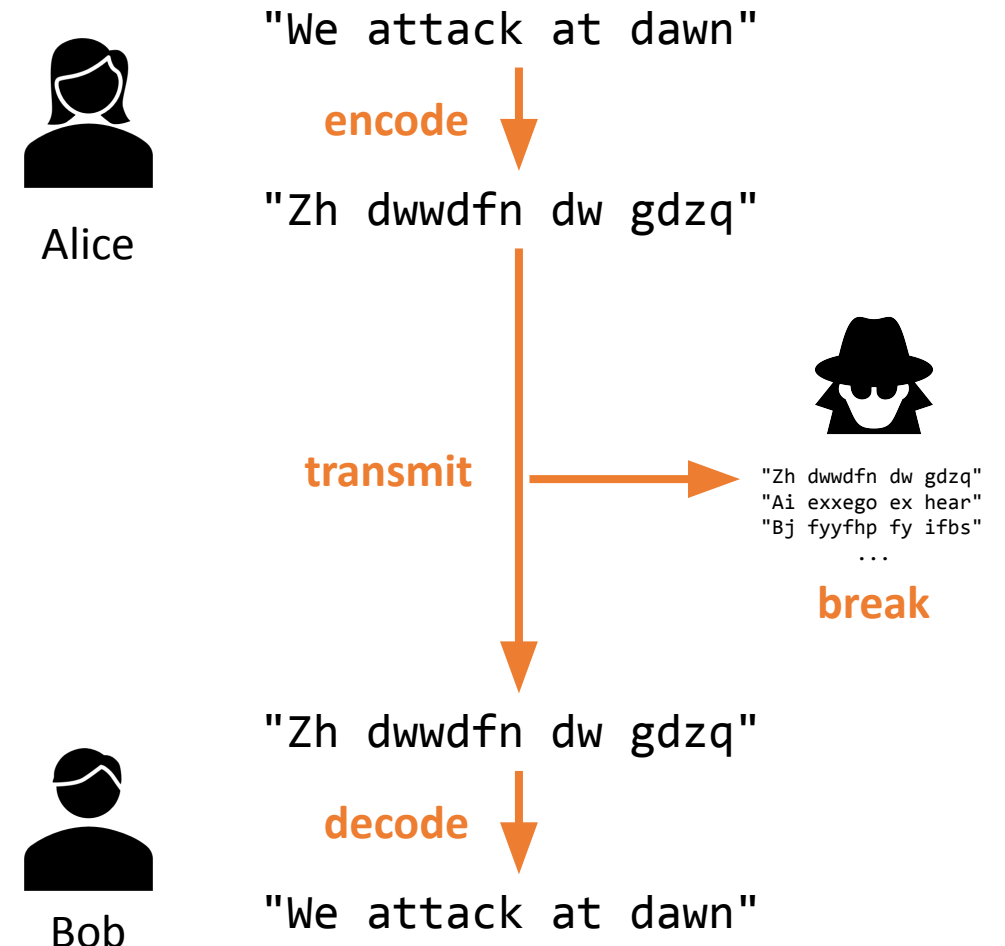
**plaintext:** actual message

**ciphertext:** obfuscated version of the message

**encode:** turn plaintext data into ciphertext data

**decode:** turn ciphertext into plaintext data

**breaking encryption:** decoding ciphertext as a third party after intercepting the message



There are **many different algorithms** that can be used for encryption, we'll talk about symmetric and asymmetric encryption.

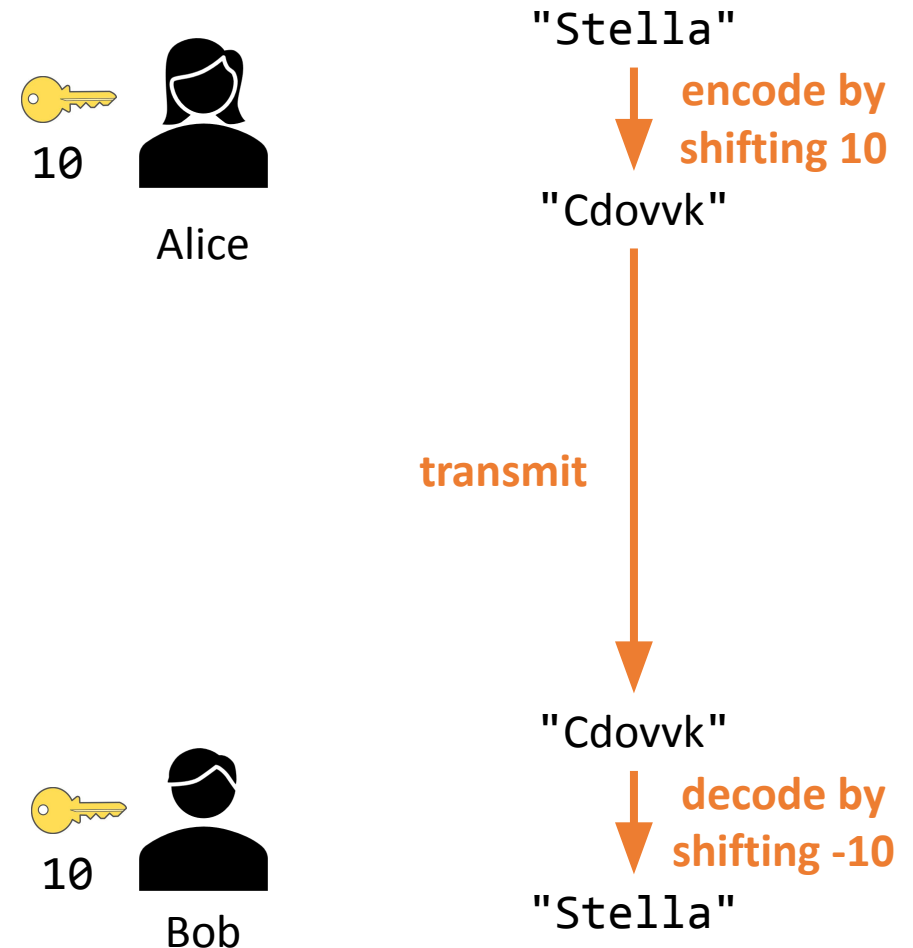
We assume that the algorithm itself is public knowledge, so we need a **secret key** to ensure that other people can't decode the message. Breaking a code usually involves determining what the key is; the stronger a key, the harder it is to break.

Some algorithms are **symmetric**: a single key needs to be known by both parties before messages can be exchanged  
Others are **asymmetric**: each person has their own key.

A **Caesar Cipher** is a simple example of symmetric encryption that uses a number between 0 and 25 as the key.

To encrypt a message, you **shift the letters in the message** by the amount specified by the key. For example, if the shift is 1, "A" becomes "B", "B" becomes "C", etc.; "Z" will become "A", as the shift wraps around.

To decrypt, simply shift the same number of letters backwards.



Can an adversary easily **break the Caesar Cipher** if they really want to read the message?

The adversary could attempt to decode the message without the key by trying to decode it with every possible key. If one of the resulting messages looks sensible, it's probably the plaintext.

**You do:** how many possible keys are there?

A small number of possible keys, makes it each to check with a brute-force approach.

There are only 26 keys – a **constant** number. That's easy to check with a brute-force approach.

Better algorithms will use a key that can **grow to any size**, like a regular password. That will make it much harder to guess the correct key!

Decode "Cdovvk":

```
"Depwwl"    "Pqbiix"  
"Efqxxm"    "Qrcjjy"  
"Fgryyn"    "Rsdkkz"  
"Ghszzo"    "Stella" <- found it!  
"Hitaap"    "Tufmmb"  
"Ijubbq"    "Uvgnnc"  
"Jkvccr"    "Vwhood"  
"Klwdds"    "Wxippe"  
"Lmxeet"    "Xyjqqf"  
"Mnyffu"    "Yzkrrg"  
"Nozggv"    "Zalssh"  
"Opahhw"    "Bcnuuj"
```

We can come up with better keys, but the two parties have to somehow **share the key**.

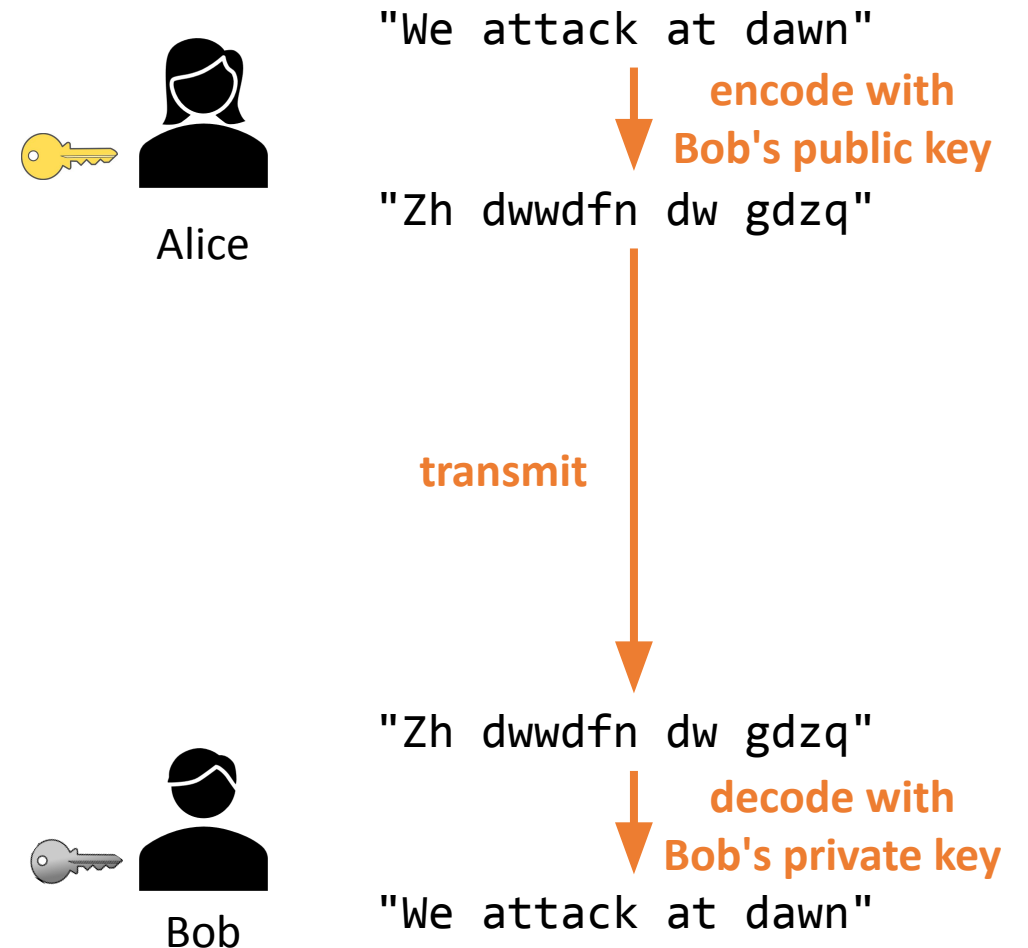
What if you want to send an encrypted message to someone you've never talked to before (like a new website)? How can you securely establish a shared key?

Shared keys are normally established by first using **asymmetric encryption**.

In asymmetric encryption each party has two keys: **a public key and a private key.**

The **public key is used for encoding** and is listed publicly where everyone can see it. The **private key is used for decoding** and is kept hidden safely away.

If Alice wants to send a message to Bob, she encodes it using **Bob's public key**. When Bob receives the message, he decodes it using **Bob's private key**.





**RSA** is a classic asymmetric encryption algorithm that is used commonly for secure communication online.

It stands for Rivest-Shamir-Adleman, the three inventors of the algorithm.

RSA encrypts messages by using mathematical operations and extremely large, hard-to-guess numbers. Guessing a key is in NP but not P.

For more info, read:

[https://en.wikipedia.org/wiki/RSA \(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

## Activity: Understanding Asymmetric Encryption

**You do:** If Bob wants to send a message to Alice, what does he do?

**A:** Bob uses **Bob's** private key to encrypt and Alice uses **Bob's** public key to decrypt.

**B:** Bob uses **Bob's** public key to encrypt and Alice uses **Alice's** private key to decrypt.

**C:** Bob uses **Alice's** public key to encrypt and Alice uses **Alice's** private key to decrypt.

# Learning Goals

- Recognize and define the following keywords: **fault tolerance**, **bottlenecks**, **net neutrality**, **data privacy**, **data security**, **DDOS attacks**, and **man-in-the-middle attacks**
- Recognize and define common approaches of **authentication**, including **passwords** and **certificates**
- Recognize and define the core elements of **encryption**, including **plaintext**, **ciphertext**, **keys**, **encoding**, **decoding**, and **breaking**
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA**, and recognize whether they are **symmetric** or **asymmetric**

# Sidebar: Commonly Used Security

This is extra material; you won't be tested on it.

RSA is a classic asymmetric encryption algorithm that is used commonly for secure communication online.

RSA encrypts messages by using mathematical operations. The core idea behind RSA is that it is fairly easy to find three numbers **d**, **e**, and **n** such that:

$$(x^e)^d \bmod n == x$$

If we can translate our message into a number **x**, we can use **(e, n)** as the public key and **(d, n)** as the private key.

# RSA Encryption/Decryption Steps

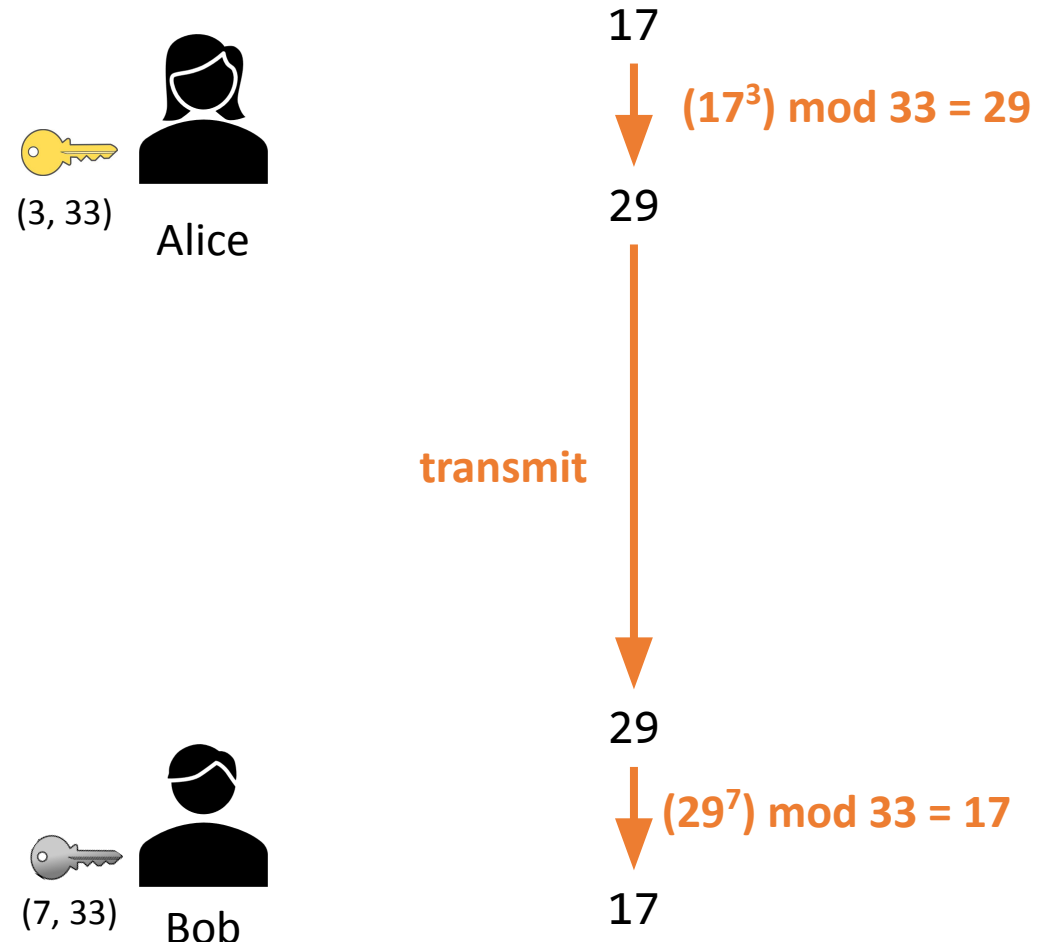
First, translate the message into a number. Let's say our message translates into the number 17.

Next, find the receiver's **public key** listed online. This is the pair  $(e, n)$ . Let's say our receiver has generated the set of numbers  $(d=7, e=3, n=33)$ ; we receive  $(3, 33)$ .

Encode the message by evaluating  $x^e \bmod n$ . That gives us the ciphertext,  $c$ .

Send the message to the receiver. They use their **private key**  $(d, n)$  to finish the equation, by computing  $c^d \bmod n$ . This is equivalent to  $(x^e)^d \bmod n$ , which is  $x$ .

Once they translate that number back to text, they can read the original message.

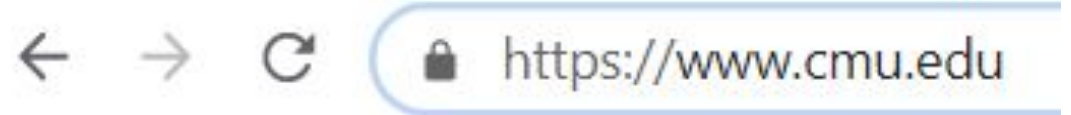


# HTTPS

We discussed in a previous lecture how the HTTP protocol makes it possible for computers to send webpages across the internet.

The **HTTPS** protocol is HTTP, but **secure**. It **encrypts** all the data included in packets so that only the sender and receiver can read it.

You can tell whether you're using HTTPS by looking at the beginning of your URL and/or checking whether there's a lock before the URL in your browser window.



# VPN

If you want to make sure your communication on the internet is both secure and private, you might use a **Virtual Private Network (VPN)**.

This is an application that creates a secure, encrypted connection between your computer and another computer (managed by the VPN) across the internet.

The VPN computer is listed as the sender and receiver of your messages; that keeps your own identity secret.





# VPN Process

To keep your internet activity **private**, a VPN uses a process called **tunneling**. The VPN application places a message inside a wrapper that disguises it to look innocent to surveillance, criminals, or content restrictions. When the message reaches the VPN computer, it 'unwraps' the message and sends the contents on to the true recipient with the VPN listed as the sender. When it gets the response, it wraps the contents and sends it back to the user.

[Tor](#) is a particularly well-known VPN service. It provides multiple layers of wrapping, so it is known as the 'Onion Router', as onions also have layers.

