Authentication and Encryption

15-110 – Friday 10/29

Announcements

- Check5 due **Monday**

- Can do most of Hw5's written component after today's lecture too!

- General reminder: don't come to class when you're contagiously sick!
  - Lecture: email the profs, we can give you the Zoom link so you can attend remotely, or you can watch the recording
  - Recitation: there's a special remote recitation scheduled as needed, contact us to request it when sick

Learning Goals

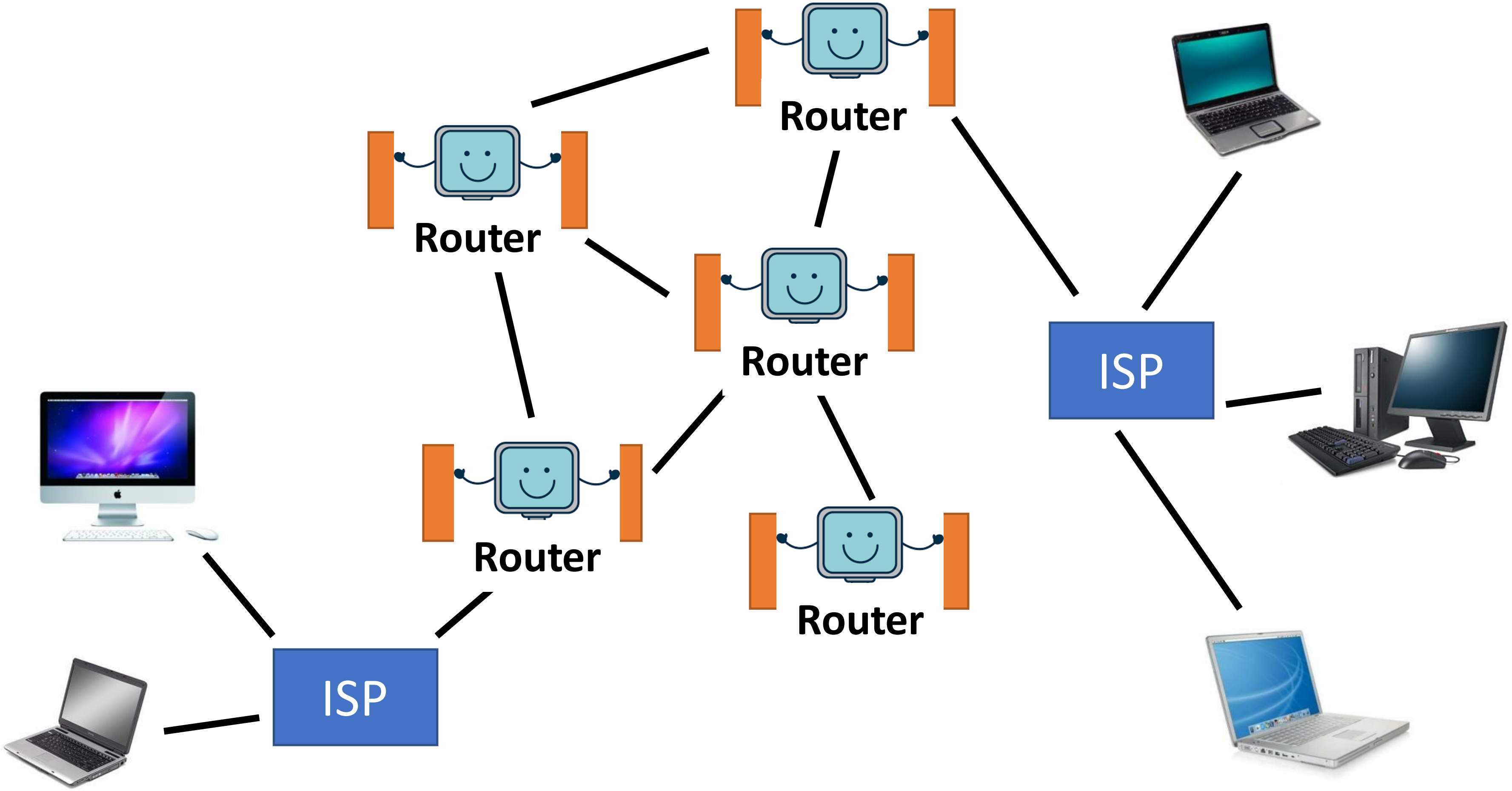
- Define the following terms: **data privacy, data security, authentication,** and **encryption**

- Recognize the traits of the internet that make it more prone to **security attacks** and recognize common security attacks (**DDOS** and **man-in-the-middle**).

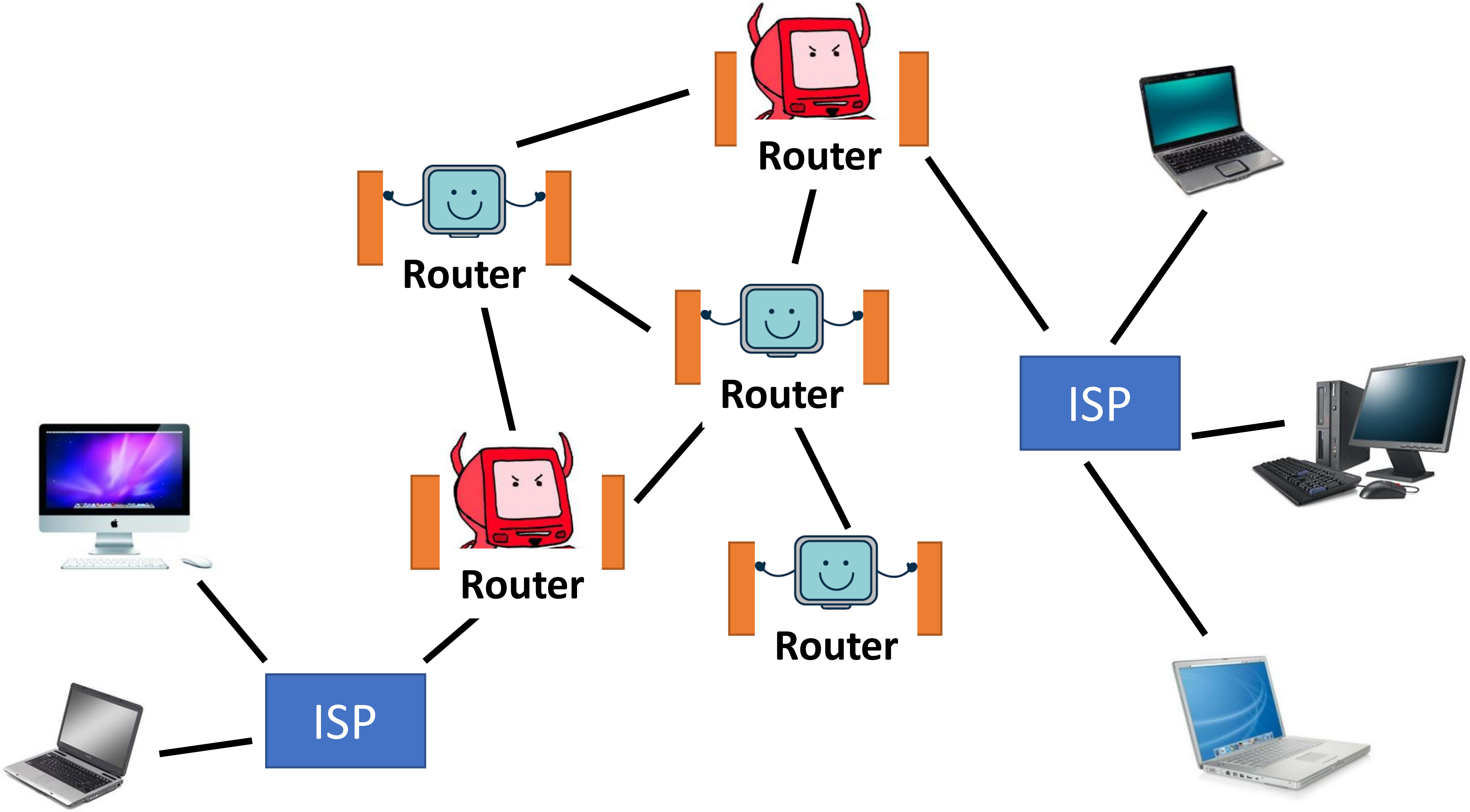
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA,** and recognize whether they are **symmetric** or **asymmetric**

- Evaluate the efficiency of **breaking** encryption algorithms based on **keyspace**.

The Internet: A Utopian Vision



The Internet: Reality



Data Privacy and Security

Defining Privacy and Security

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.

Both share a common goal: no third party should be able to read certain types of data being sent across the internet.

Adversaries Attempt to Collect Data

Adversaries are people on the internet who try to collect data that others want to keep private or secure.

They 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.

Internet Weaknesses

Security is a problem in real life too, but the internet has three characteristics that make attacks more 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's easy to distribute security vulnerability code to other adversaries

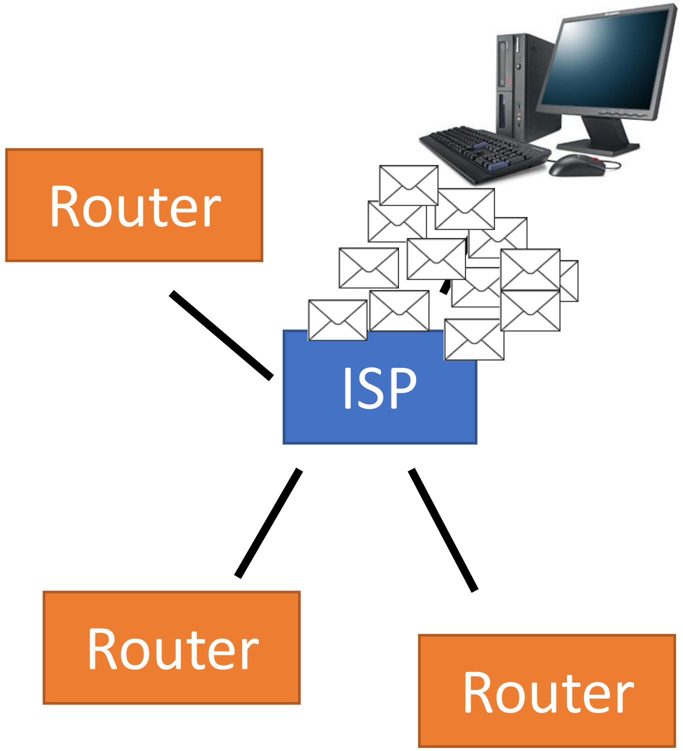
Example: DDOS Attack

One common method used by adversaries is a **Distributed Denial of Service (DDOS) attack**. Adversaries send or receives a huge amount of data to/from 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.



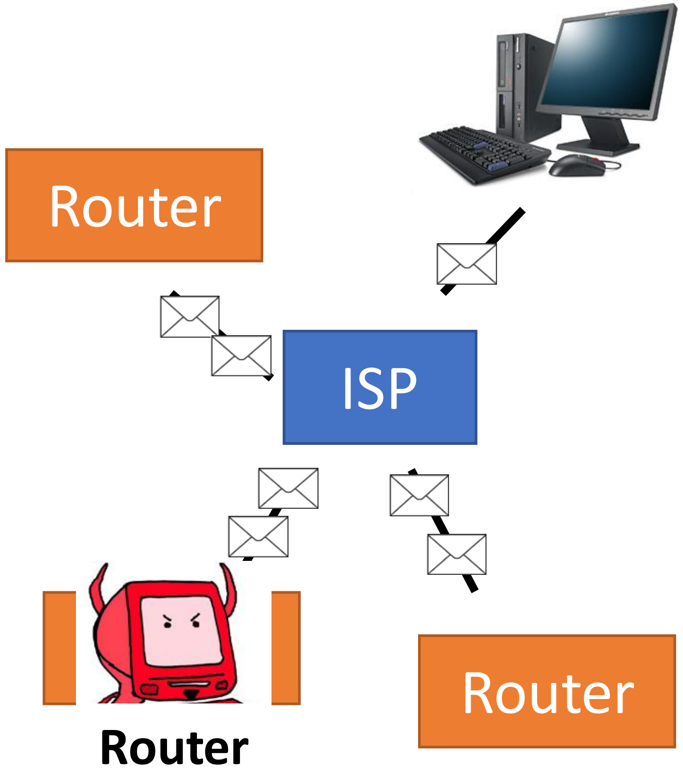
Example: Man-in-the-Middle Attack

Another common attack is a **man-in-the-middle attack**. 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 to something different, since the packets use standard protocols.

These attacks are only possible if the packets are not **encrypted** (we'll talk more about this next). They occur mostly on public, unencrypted Wi-Fi networks, like coffee shops. Organizations can also use them to track internet activity on company Wi-Fi.

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



Authentication and Encryption Protect Data

Luckily, we have algorithmic ways to protect our data from adversaries.

Today we'll talk about two main approaches: **authentication** (which is used for verification) and **encryption** (which is used for obfuscation).

Authentication

Authentication Confirms Identity

Authentication is a process that confirms someone is who they say they are. It's 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.

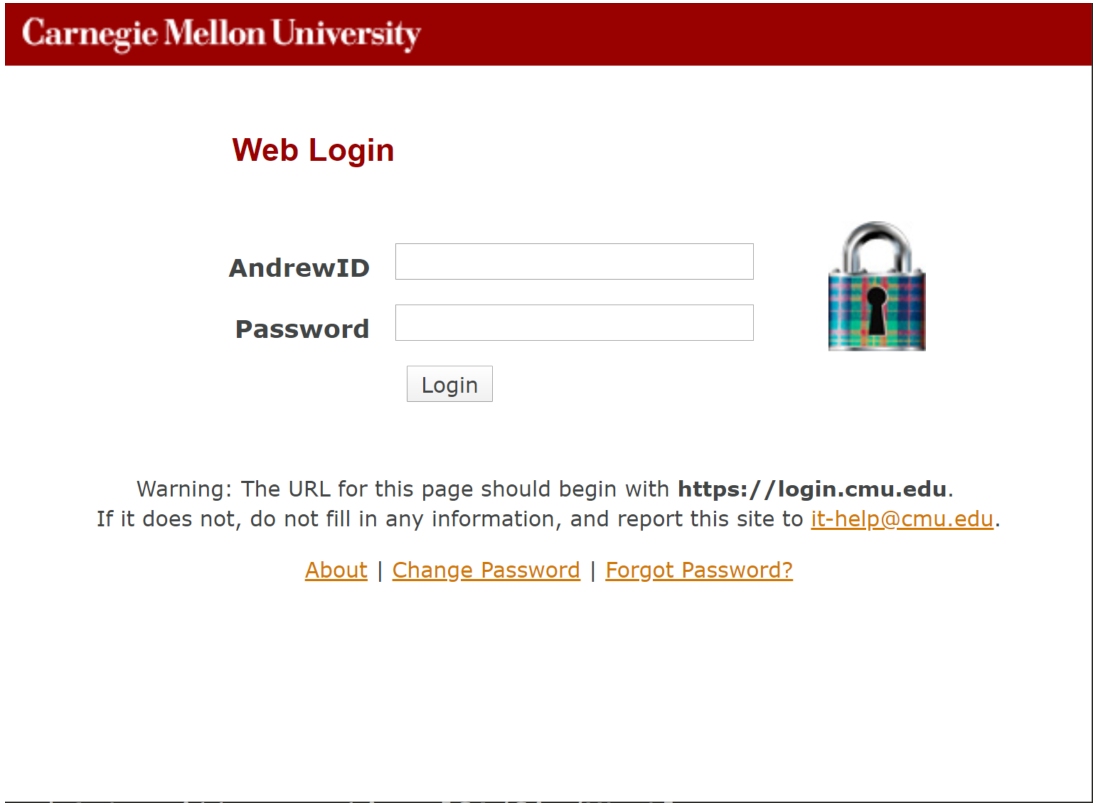
Websites also authenticate their identity during secure online communications.

Authentication via Password

To verify that a person is who they say they are, we often use **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.

Passwords can be text, biometric data, or even 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.

Verifying Passwords

When you enter your password into a login page, the password is **encrypted** before being sent across the internet to the website.

The website will (hopefully) have your password stored in its encrypted format in a database on its server. The server can just check whether the two encrypted strings are the same.

What happens if the server stores the decrypted version instead? If an adversary gets access to the database, they get all the passwords!

Guessing a Password – Algorithm?

If an adversary wants to steal authentication, they need to figure out what the password is. How can that be done?

They could use a **brute-force approach**, but we know that those aren'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!

Guessing a Password – Reality

More often, adversaries use **human vulnerabilities** to guess passwords. This includes **dictionary attacks**, where they guess dictionary words (as users often just use a real word as a password). It also includes **social engineering**, where the adversary tries to trick someone into revealing their password.

Adversaries also use **security vulnerabilities** in websites (human or technical) to access user data. 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.

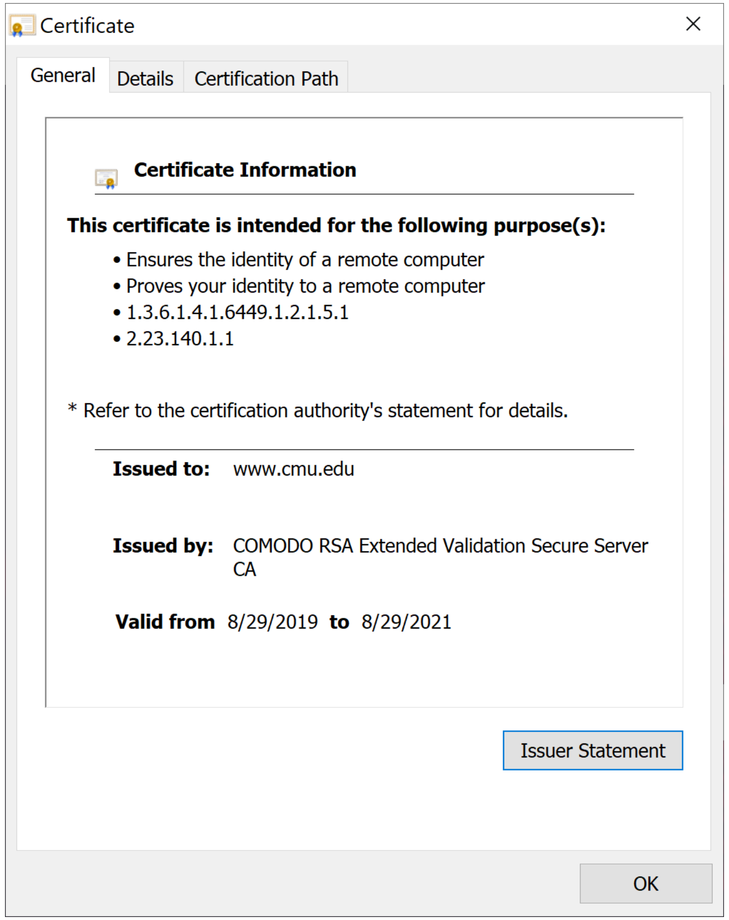
If you have a password that is not a dictionary word, is reasonably long, and isn't used on other websites, it is very hard for someone to "hack" your account.

Authentication via Certificate

Websites also need to verify their identities on the internet. For example, when you log in to your bank account, you want to make sure it's actually your bank.

A **certificate** is data that confirms the website that holds it is the equivalent of a real-world organization. You can view a website's certificate by clicking on the lock icon to the left of the URL in your browser.

Certificates are sent along with website data whenever you do encrypted communication with a website.



Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- Proves your identity to a remote computer
- 1.3.6.1.4.1.6449.1.2.1.5.1
- 2.23.140.1.1

* Refer to the certification authority's statement for details.

Issued to: www.cmu.edu

Issued by: COMODO RSA Extended Validation Secure Server CA

Valid from 8/29/2019 **to** 8/29/2021

Certificate Authorities Issue Certificates

Certificate 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. If a certificate authority starts doing a bad job at verifying identities, it will usually be removed from those lists.

You do: Real Life Authentication

You do: we use authentication in real life too. What is an example of a situation where you need to authenticate your identity in real life?

Are there also situations where companies need to authenticate *their* identities in real life?

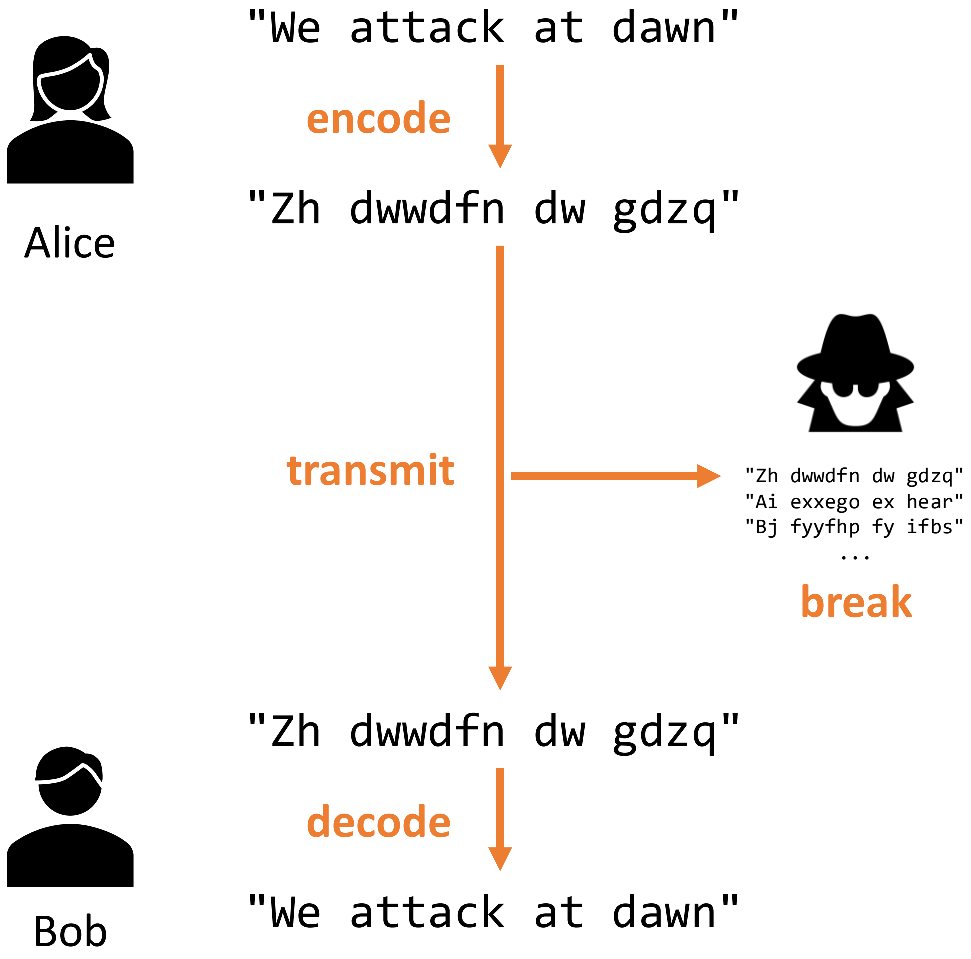
Encryption

Encryption Encodes Data

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

When working with encryption, we often refer to two types of data: the **plaintext**, which is the actual message, and the **ciphertext**, which is an obfuscated version of the message.

We'll talk about **encoding** data, **decoding** data (undoing an encryption as the receiver), and **breaking** an encryption (decoding it as a third party after intercepting the message).



Encryption Algorithms

There are many different algorithms that can be used for encryption. We'll discuss the **Caesar Cipher** and **RSA** here.

For most encryption algorithms, we assume that the algorithm itself is public knowledge. That means that 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.

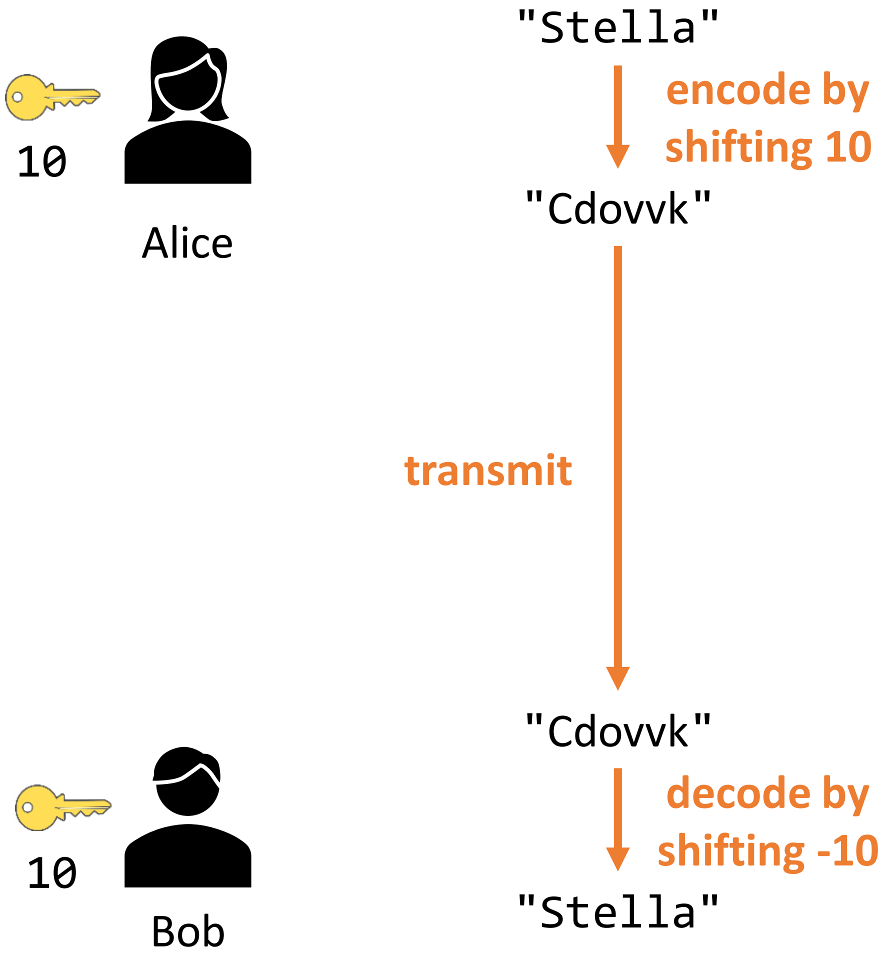
Some algorithms are **symmetric**; that means that a single key needs to be known by both parties before messages can be exchanged. Others are **asymmetric**; each person has their own key.

Example: Caesar Cipher

The **Caesar Cipher** is a very simple encryption algorithm. It is **symmetric**, so it uses a single key known by both parties. That key is some number between 0-25.

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.



Breaking Caesar Cipher

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?

Keyspace Determines Algorithm Strength

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

The **keyspace** of an algorithm is the number of possible keys that can be used to encrypt a message. Traditionally we refer to the size of a keyspace as the number of **bits** it takes to represent all possible keys. If a key is k bits long, there are 2^k possible keys for the adversary to check.

For Caesar Cipher, we only need 5 bits to represent the key ($2^5 = 32$), so the keyspace is 2^5.

```
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"   "Abmtti"
           "Bcnuuj"
```

Runtime to Break Caesar Cipher

If there are 2^5 possible keys for Caesar Cipher, what is the worst-case runtime to **break** a cipher?

For breaking, we don't care about the length of the message- we care about the length of the potential **key,** in terms of **bits**. Can we grow the size of the key, or not?

Caesar Cipher has a **constant** number of bits used to represent the key. That means breaking the Caesar Cipher algorithm is **O(1)**; it doesn't get more difficult with longer keys, because there's a limit on how big the key can be.

Problem: How to Share Keys?

We can design symmetric encryption algorithms that are stronger than the Caesar Cipher, with keys that grow with the size of the input, and real systems use these stronger algorithms. However, they still rely on the two parties having a **shared 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?

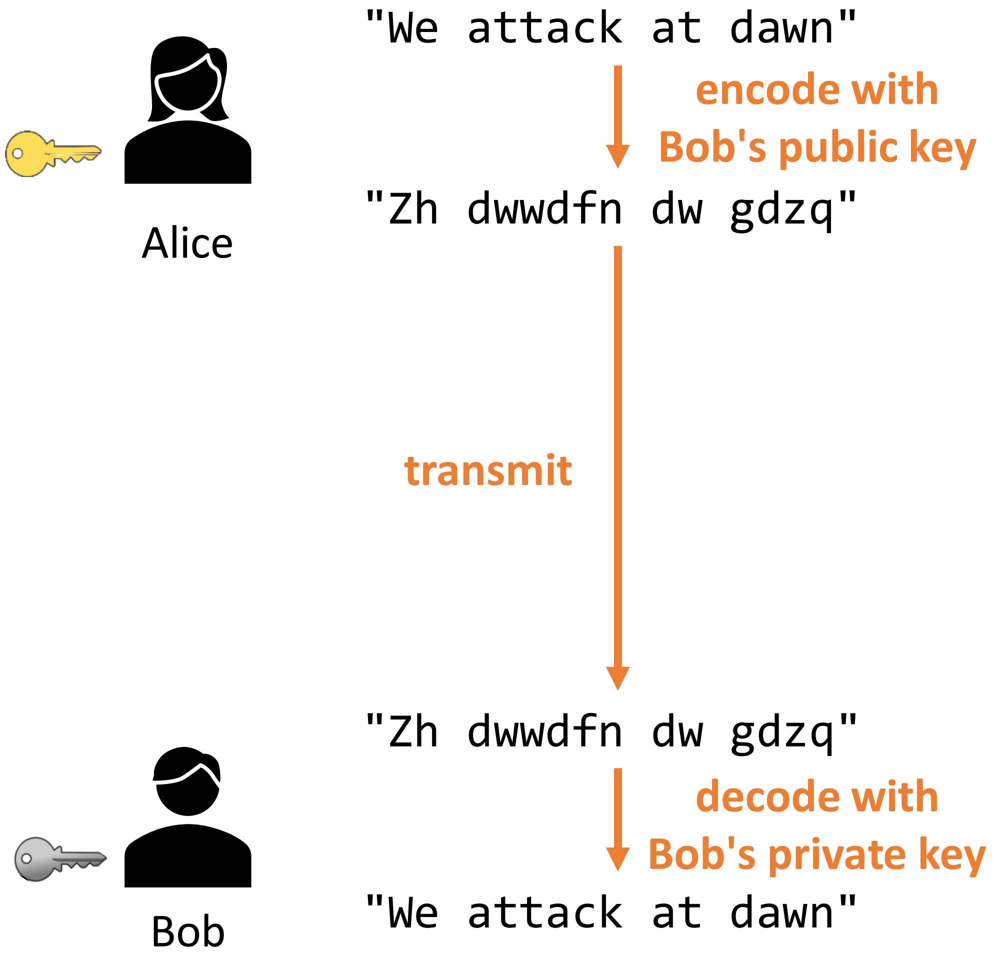
Shared keys are normally established by first using **asymmetric encryption**.

Public and Private Keys

The core idea of asymmetric encryption is this: instead of two parties holding one shared key, **every person** holds 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 **his private key**.



Example: RSA

RSA is an 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. 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

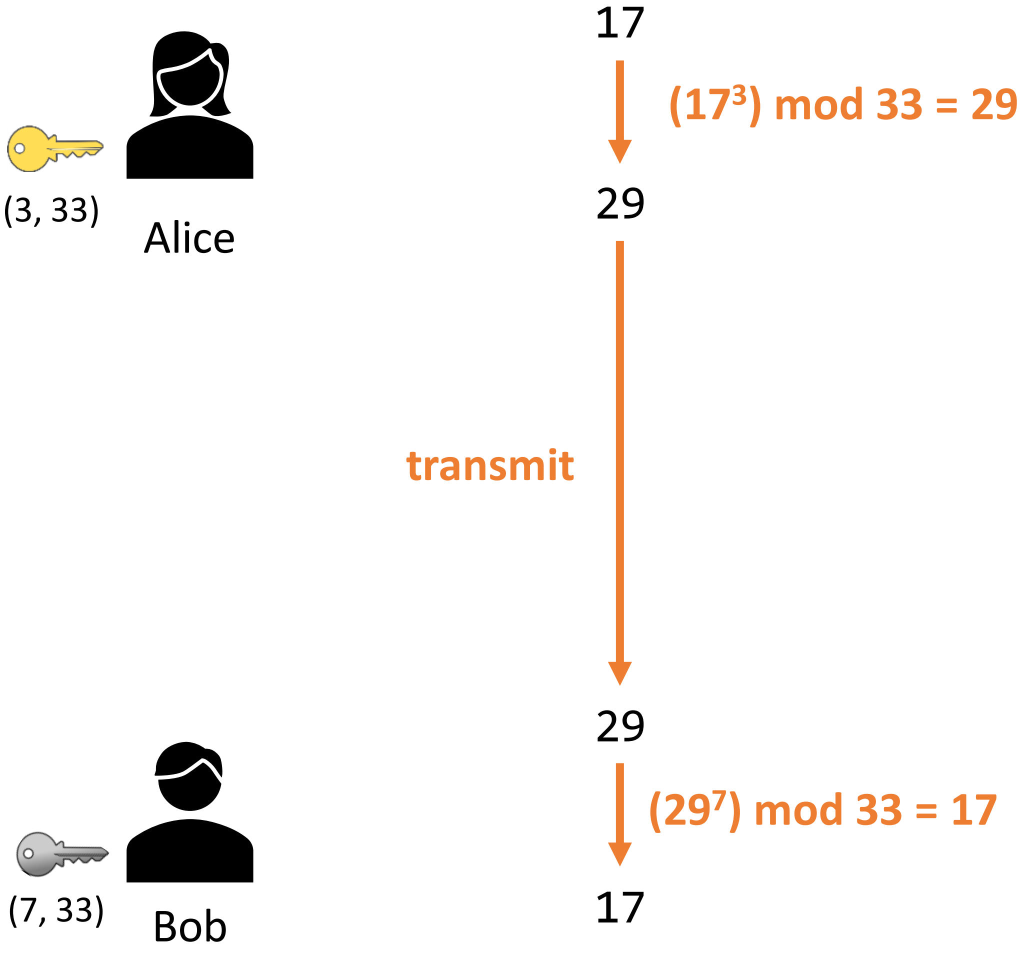
First, translate the message into a number. Let's say our message translates into the number 17.

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 mod 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 mod n. This is equivalent to $(x^e)^d$ mod n , which is x.

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



17

(17^3) mod 33 = 29

29

transmit

29

(29^7) mod 33 = 17

17

Activity: Understanding RSA

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.

Generating the Keys

How do we generate **d**, **e**, and **n** to make the public and private keys?

Use prime numbers! Find two huge prime numbers **p** and **q**, and set **n = p*q**.

d and **e** are calculated with slightly more complicated math (check Wikipedia if you're interested). What's important is that these numbers are **derived** from **p** and **q** as well.

Breaking RSA

How could an adversary break RSA? They can easily find **e** and **n** (it's public), but they'd need to determine what **d** is.

One approach is to find the numbers that generated **d; p** and **q**. That is, find the two factors of **n**, which is public. This is faster than trying every possible number in the world.

In our previous example, it's easy to determine that 33 = 3*11. But real RSA algorithms use huge prime numbers to generate an enormous **n**.

How can we find the factor pair for **n**? Iterate over **all** possible factors. Once you find the first factor, you can quickly determine the second as well. How many possible first factors are there? **Approximately n!**

RSA Keyspace

Recall that when we discuss keyspace, we care about bits. Let's say that b is the number of bits needed to represent n. The keyspace of RSA is therefore **2^b**, since there are ~n possible factors to check.

What's the worst-case runtime to break RSA? It's no longer constant, because we can now **grow** the number of possible keys by growing **n**. Instead, it's **$O(2^b)$**. In other words, **the more bits we add to n, the harder it is to crack RSA**.

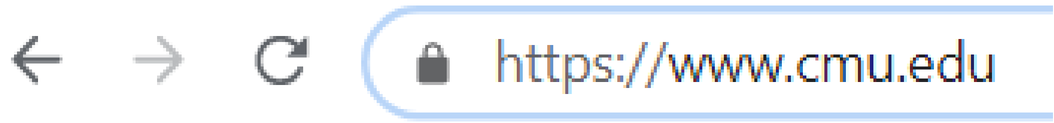
It turns out that factoring a huge number is **hard**. In fact, factorization is in NP! **This makes RSA near-impossible to break** (at least so far).

Sidebar: Security on the Internet

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.


https://www.cmu.edu

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.

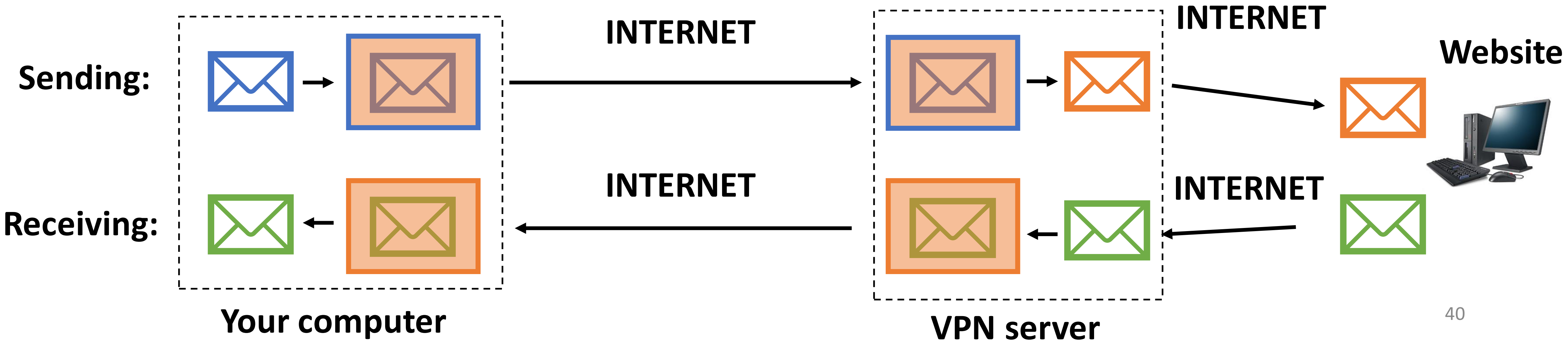
The VPN computer is listed as the sender and receiver of your messages; that keeps your own identity secret.

"On the Internet, nobody knows you're a dog."

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.



Learning Goals

- Define the following terms: **data privacy, data security, authentication,** and **encryption**
- Recognize the traits of the internet that make it more prone to **security attacks** and recognize common security attacks (**DDOS** and **man-in-the-middle**).
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA,** and recognize whether they are **symmetric** or **asymmetric**
- Evaluate the efficiency of **breaking** encryption algorithms based on **keyspace**.

Feedback: https://bit.ly/110-f21-feedback