Introduction to Computer Science I
COMP 2406 – Fall 2019

HTTPS

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Understand the difference between HTTP/HTTPS

Understand the basics of public-key encryption

Create Node.js servers that support HTTPS

The Need for HTTPS

Last lecture we discussed logging in to a web service

This is a necessary feature for many operations on the web that involve personal information and authorization

The Need for HTTPS

HTTP, however, is a plain-text protocol

Information we transfer through HTTP can easily be read by anybody observing the network data

This would make logging in with usernames/passwords pointless

The Need for HTTPS

Additionally, in basic HTTP, we have no way of verifying who we are communicating with

A 'man-in-the-middle' could intercept communications and pose as the server/client/both

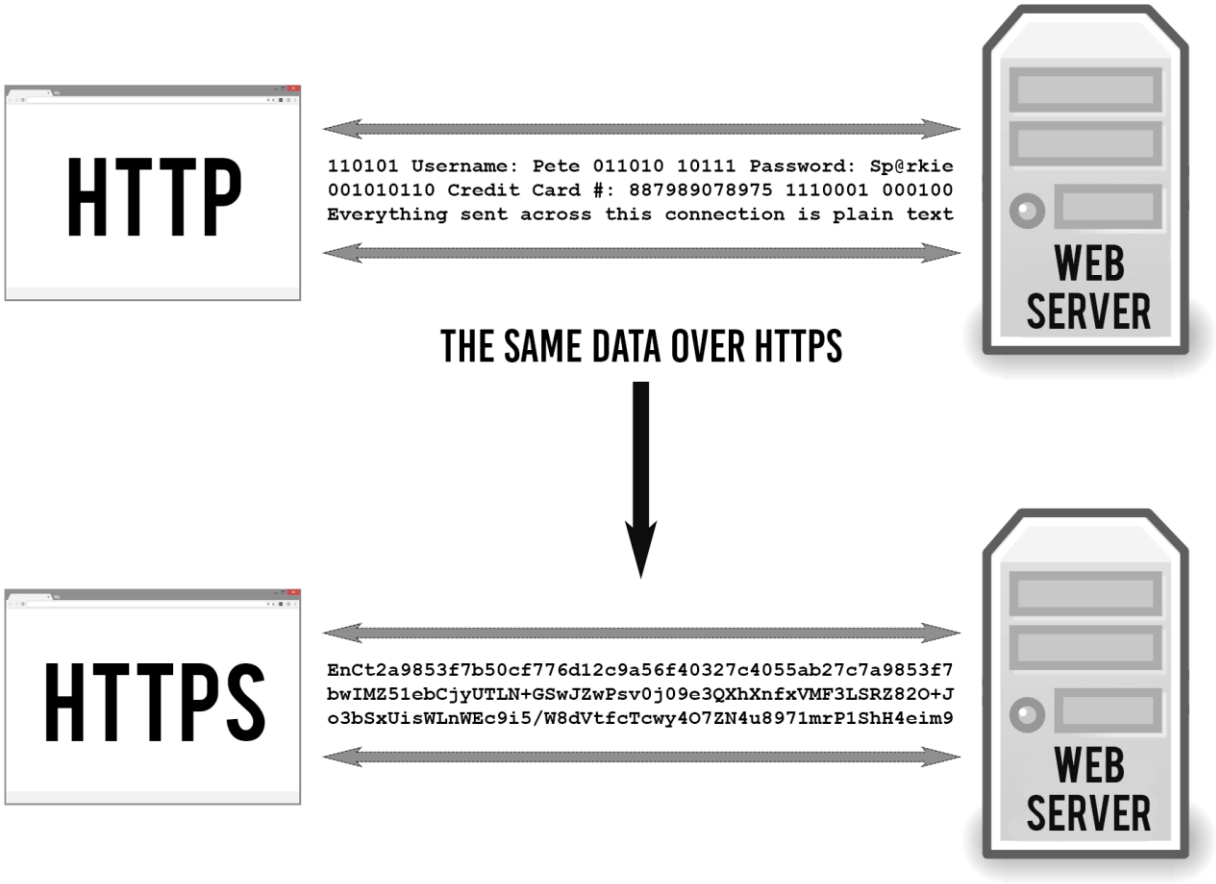
The Need for HTTPS

To solve these problems, we need to introduce encryption and authentication into the communication protocol

This is the role of HTTPS (Hypertext Transfer Protocol Secure)

HTTPS wraps the HTTP protocol with an encryption layer (Transport Layer Security, TLS)

The Need for HTTPS



https://tiptopsecurity.com/wp-content/uploads/2017/06/WhatIsHTTPSEncryption.png

The Need for HTTPS

TLS is responsible for ensuring:

1. The connection between client/sever is secure/private (encrypted)

2. The identities of the parties are authenticated (the server is who it claims to be)

3. The connection is reliable (messages cannot be tampered with)

The Need for HTTPS

TLS, and its predecessor SSL (Secure Sockets Layer) used to be used only for transactions that involved private information (payments, etc.)

Now it is becoming common for sites to use ONLY HTTPS

Many browser developers limit the functionality of sites that are operating with plain HTTP (e.g., microphone, camera, Bluetooth, etc.)

The Need for HTTPS

As we will discuss, HTTPS requires your server to have a TLS certificate

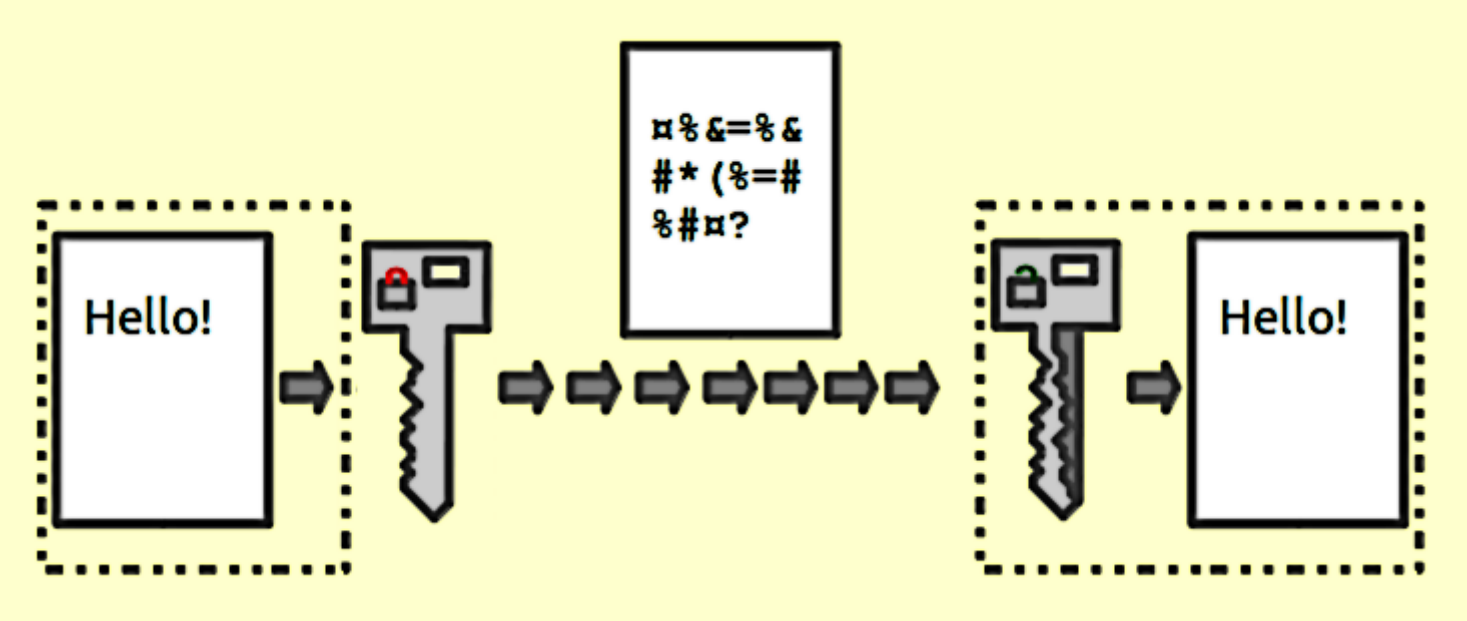
This used to be an extreme barrier – getting certificates from trusted authorities was expensive

But now free alternatives exist (Let's Encrypt)

Preliminary Information

Before we get into how encryption works within the context of HTTPS - how does encryption work in general?

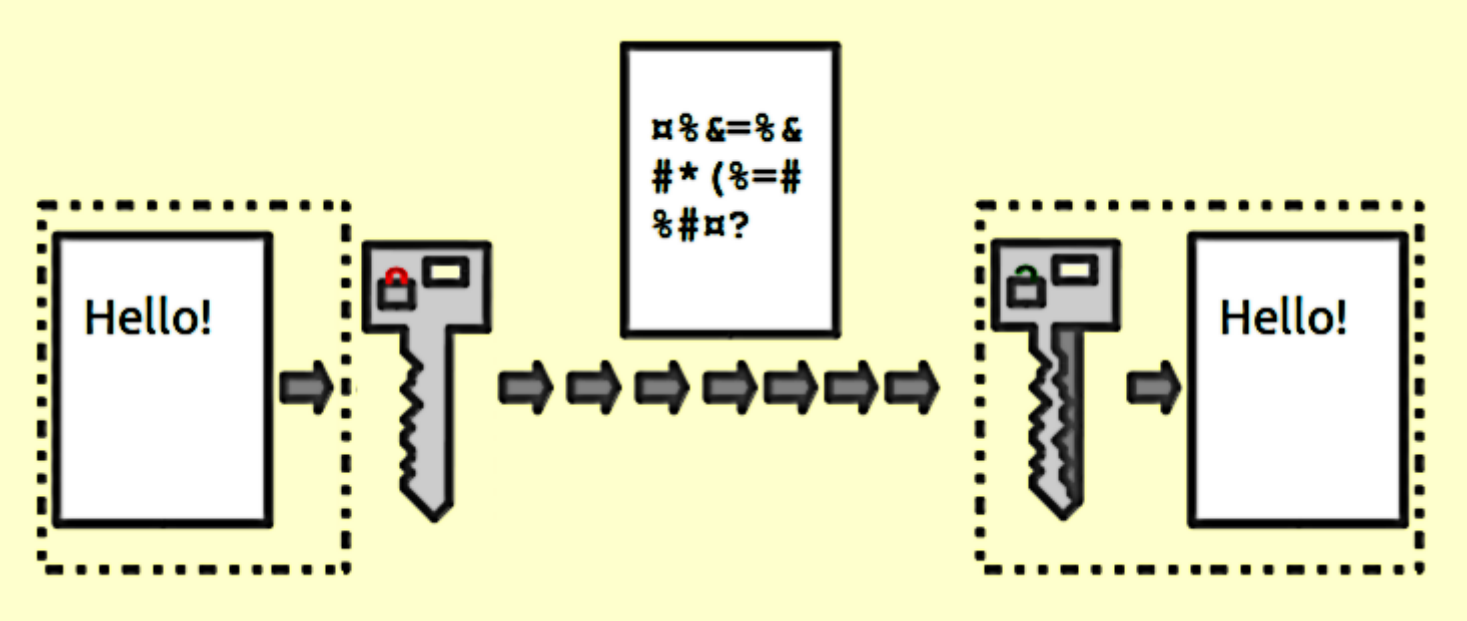
Preliminary Information



In general, you need 3 things: data you wish to encrypt (plain text), an encryption key (a long random string), and an encryption algorithms

https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537

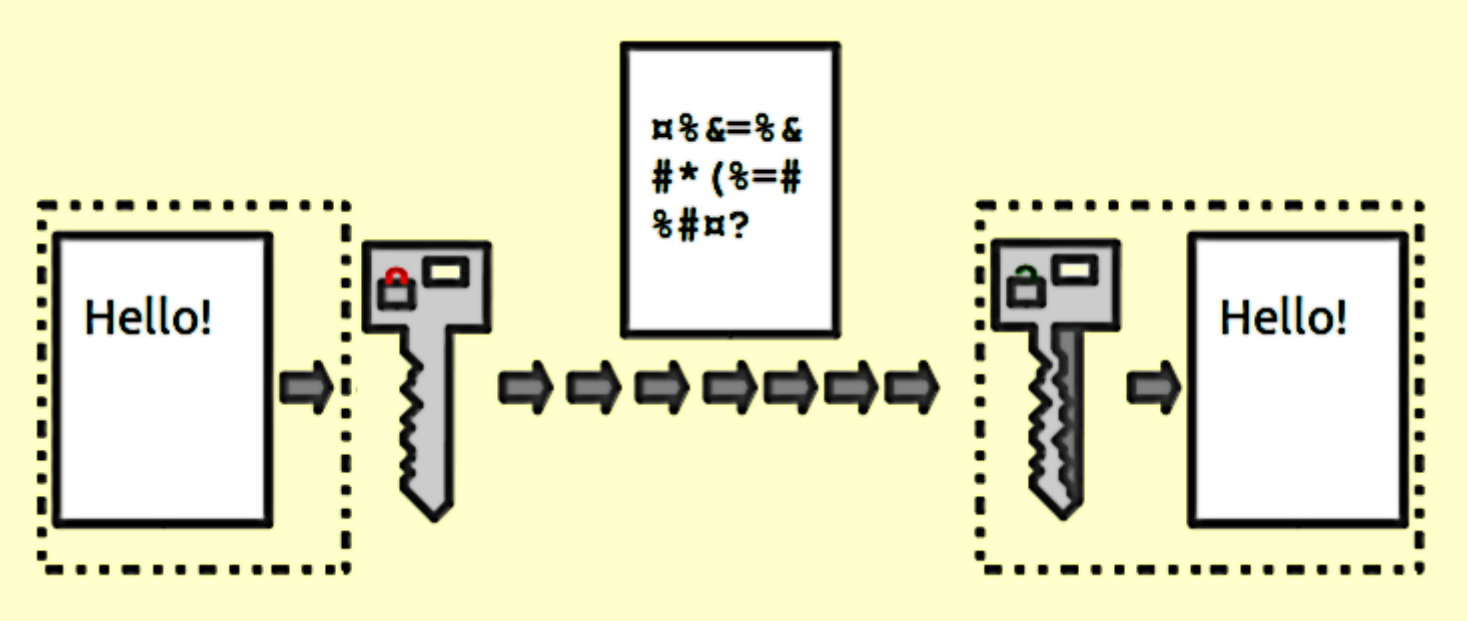
Preliminary Information



Start with some plain text. Use the encryption algorithm and key to transform the text into 'cyphertext'. Decrypt that cyphertext using the key. Without the key, the message cannot (easily) be read.

https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537

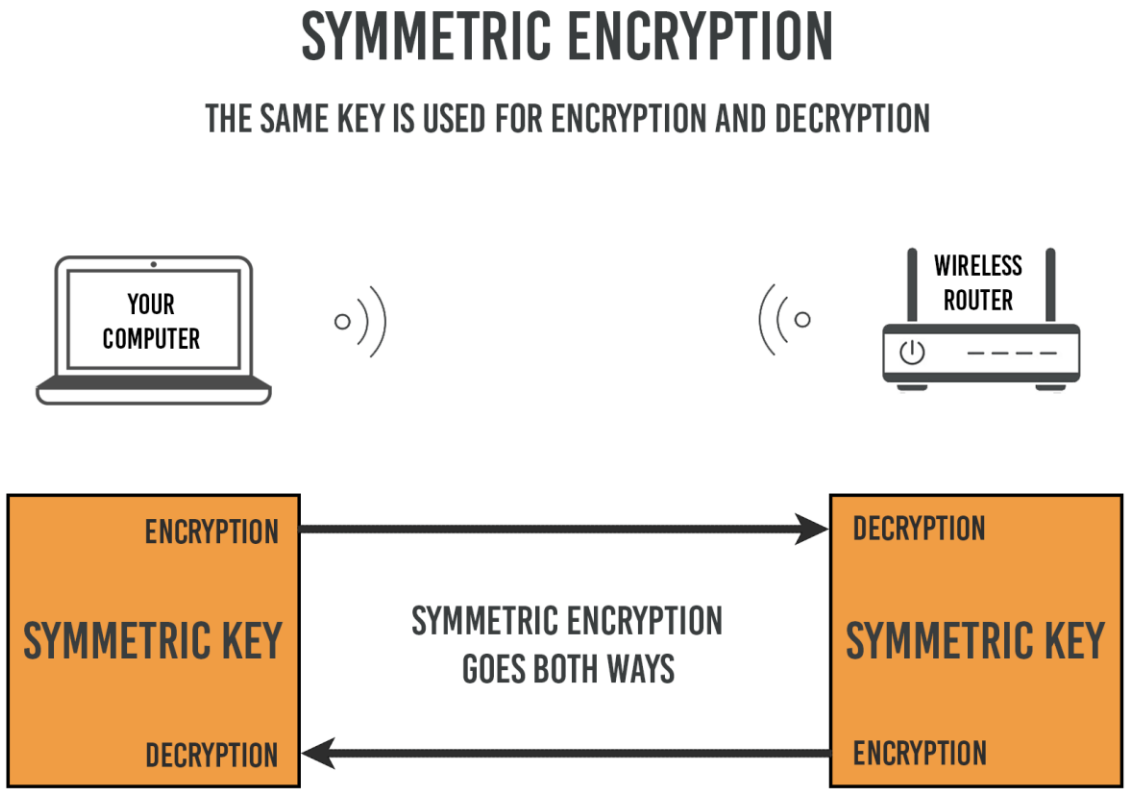
Preliminary Information



The most important part is that the key used is secret. If somebody else knows it, they can decrypt the data using the same algorithm.

https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537

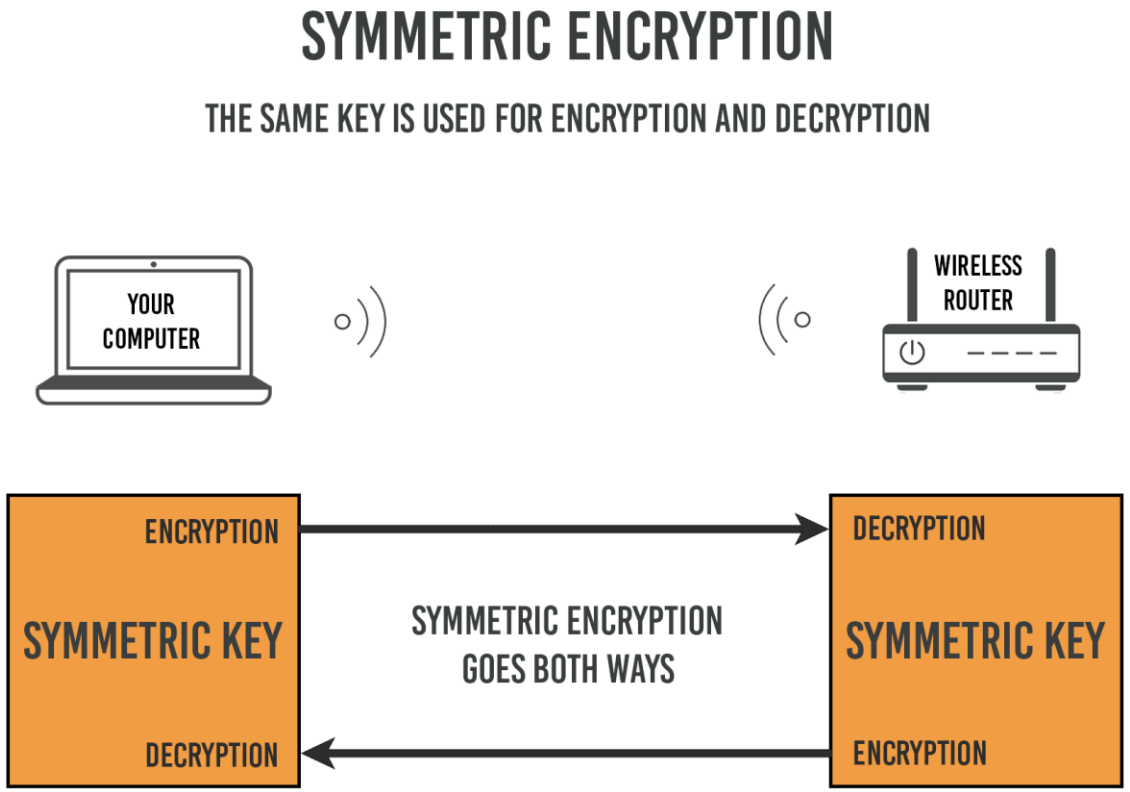
Preliminary Information



When the same key is used for encryption and decryption, this is called symmetric-key encryption

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Symmetric.png

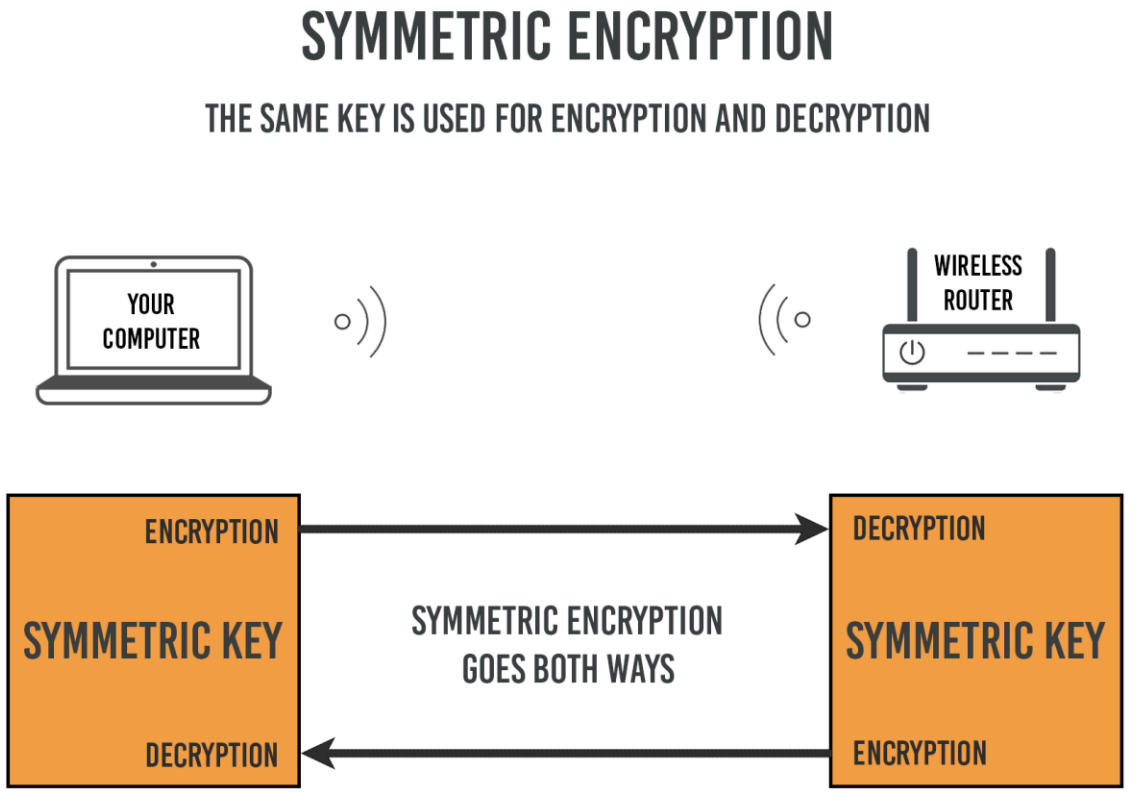
Preliminary Information



For example, your Wi-Fi router likely uses symmetric key encryption

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Symmetric.png

Preliminary Information



The router knows the key, you know the key, you tell the key to anybody you want

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Symmetric.png

The Difficulty of Secure HTTP

The design of the internet makes this more challenging

The communication network we are using to establish communication is insecure

For example, Google cannot tell you the encryption key directly, as anybody could see it and decrypt your messages

Asymmetric Encryption

So how can this be achieved?

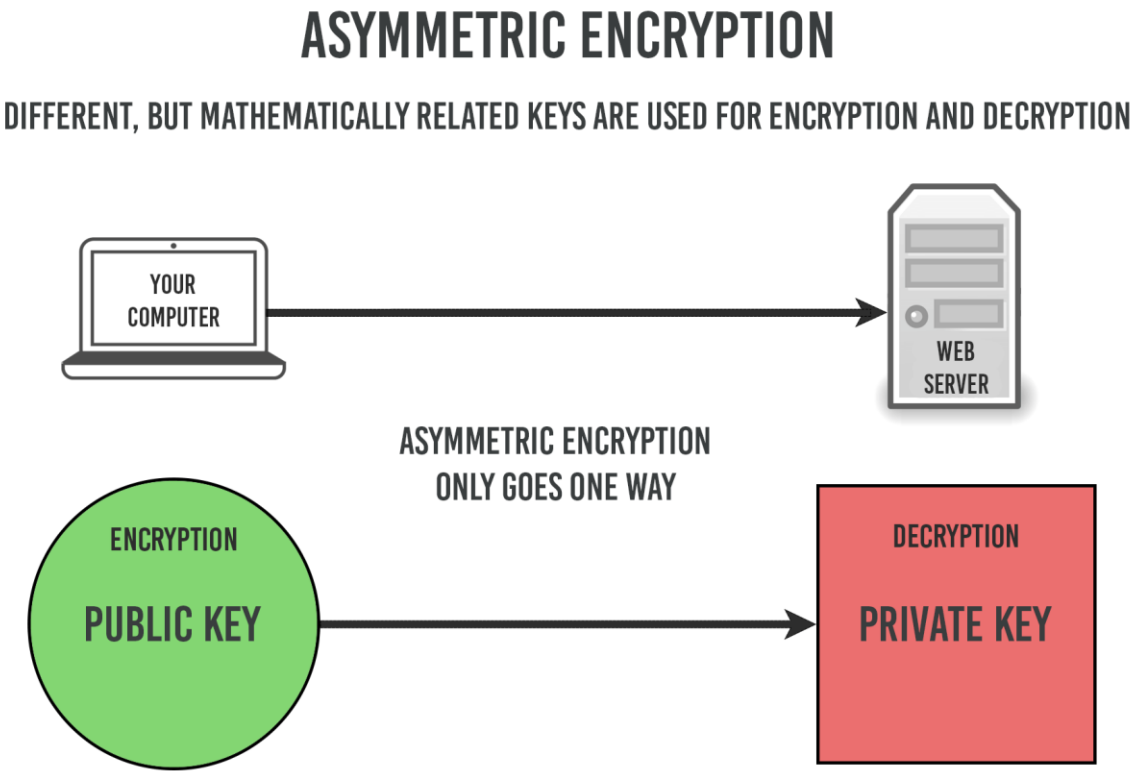
We use what is called asymmetric encryption (a.k.a. public key encryption)

This uses keys that come in pairs:

1. a public key available to anybody

2. A private key known only to one party (e.g., the server)

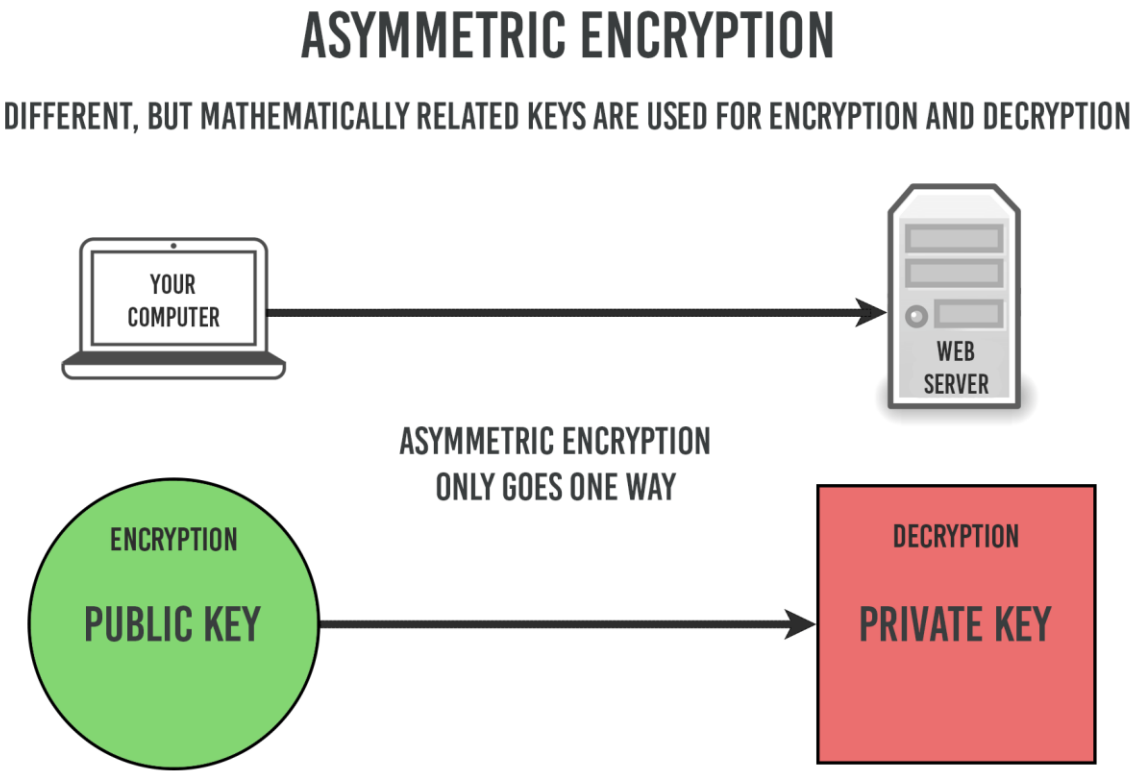
Asymmetric Encryption



The pair of keys is generated such that
1) The public key can encrypt plaintext and…

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

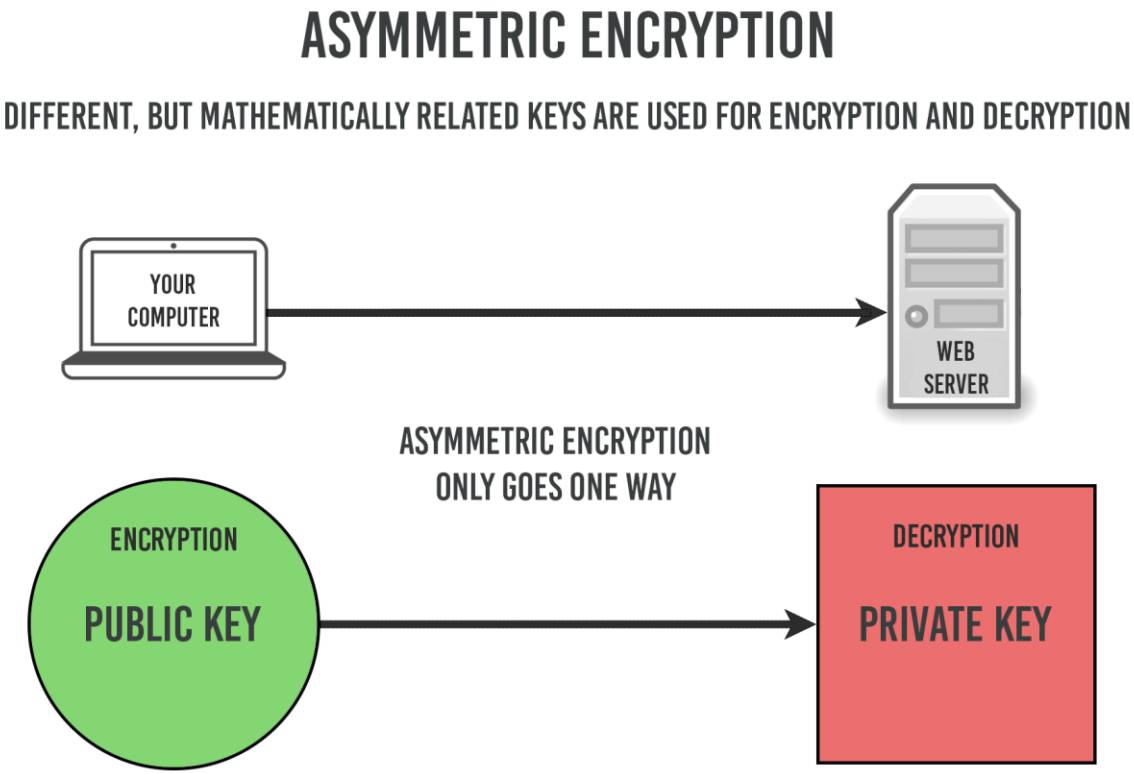
Asymmetric Encryption



The pair of keys is generated such that
2) only the private key can decrypt the cyphertext

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

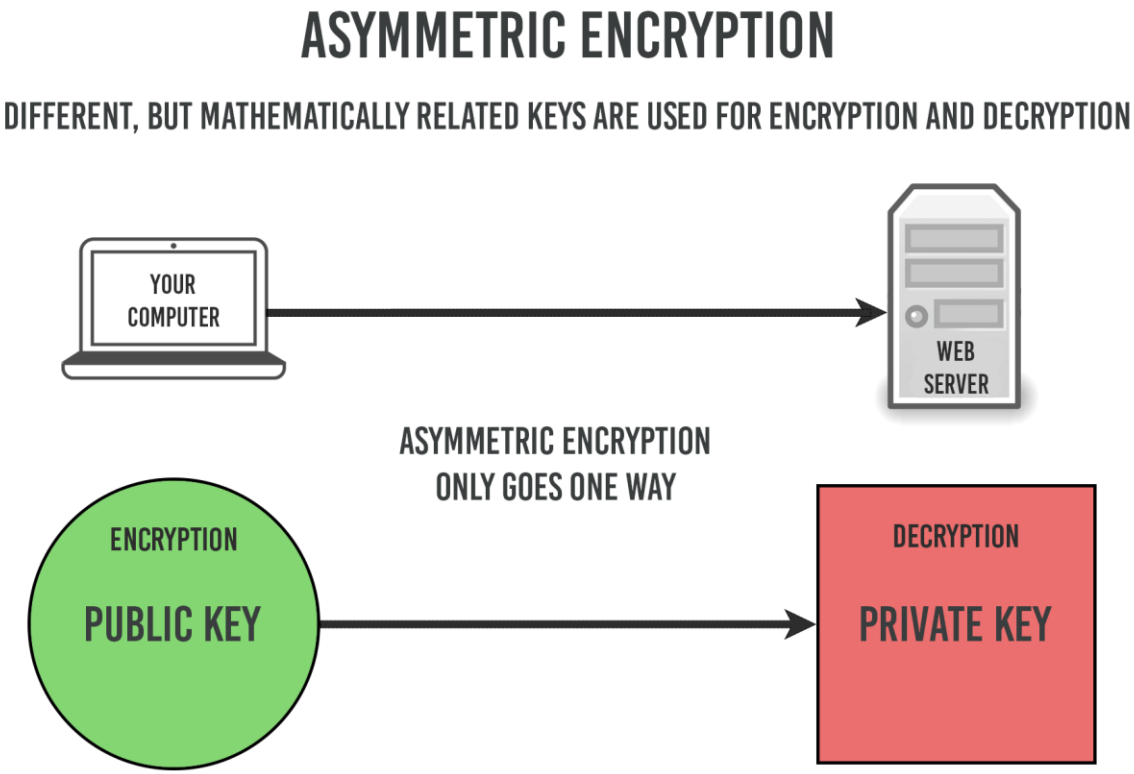
Asymmetric Encryption



If some other key is used for either encryption or decryption, it should not work (VERY small chance)

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

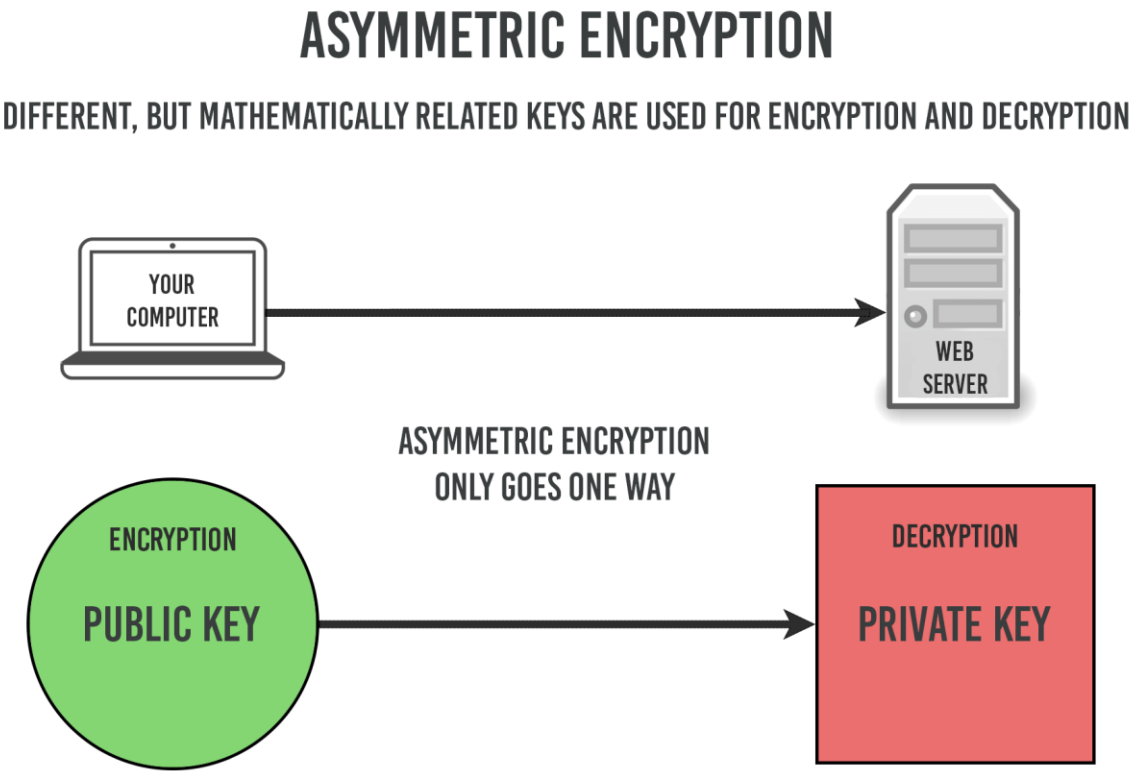
Asymmetric Encryption



This way, it does not matter who has the public key. The private key is required to decrypt the data.

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

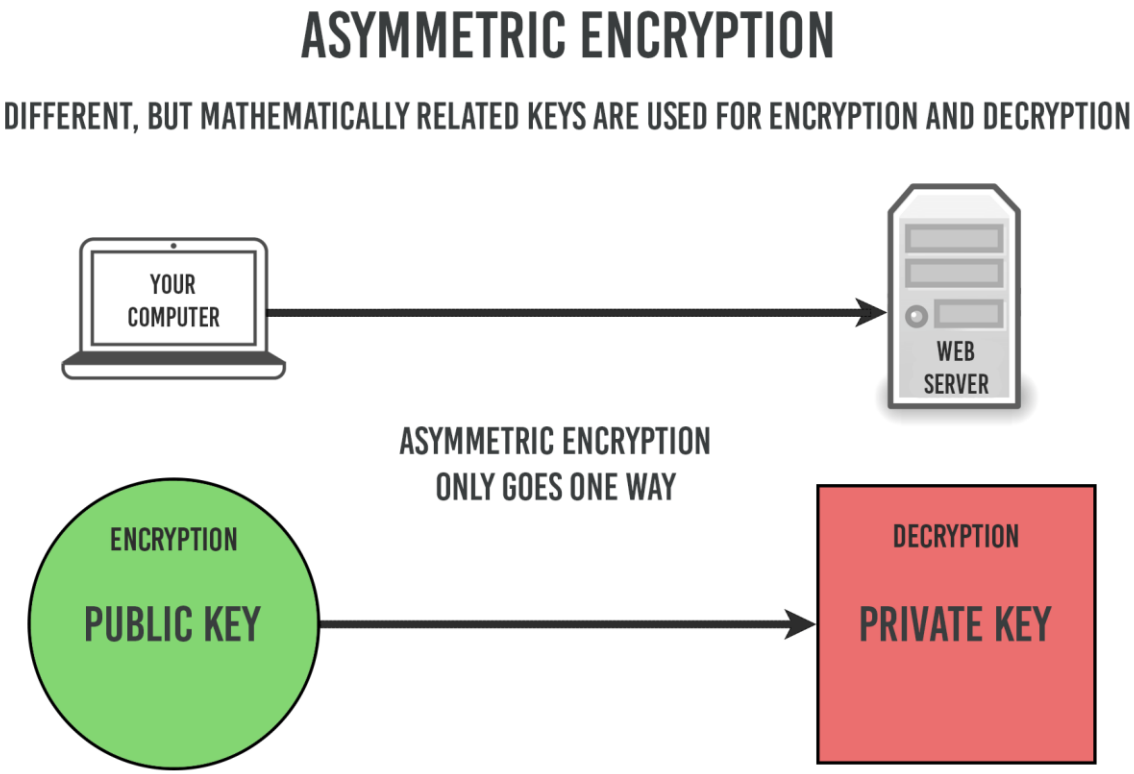
Asymmetric Encryption



So a server can share its public key and that allows clients to encrypt data only the server can decrypt

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

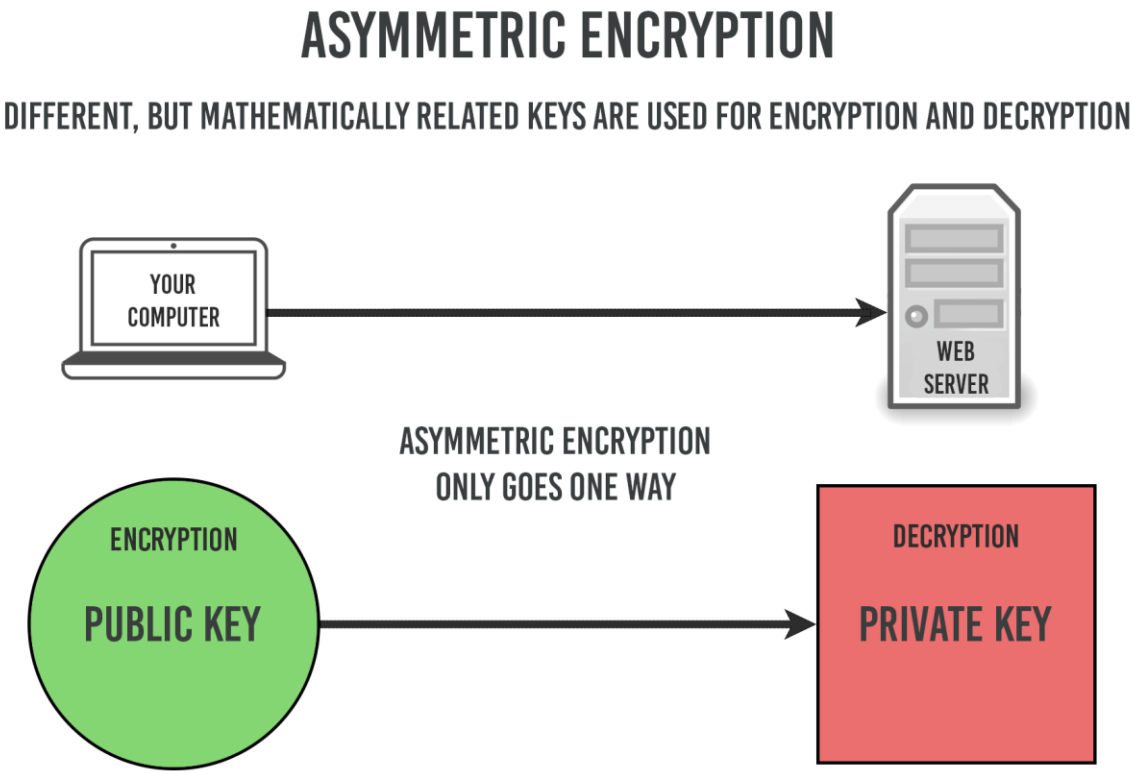
Asymmetric Encryption



But how, then, does the client read information that the server sends back?

https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

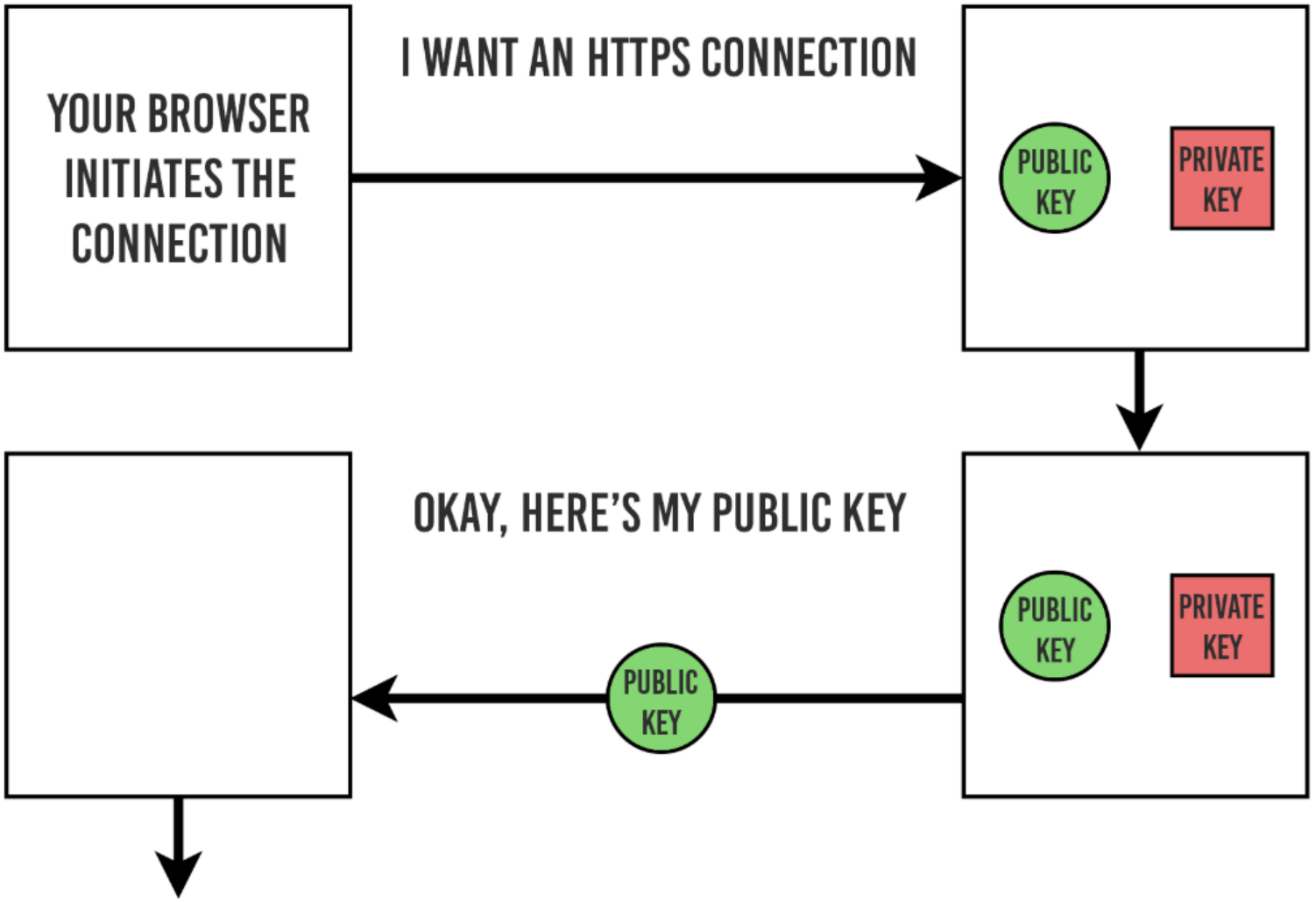
Asymmetric Encryption



They privately decide on a private key known to both parties and then use symmetric encryption. How?

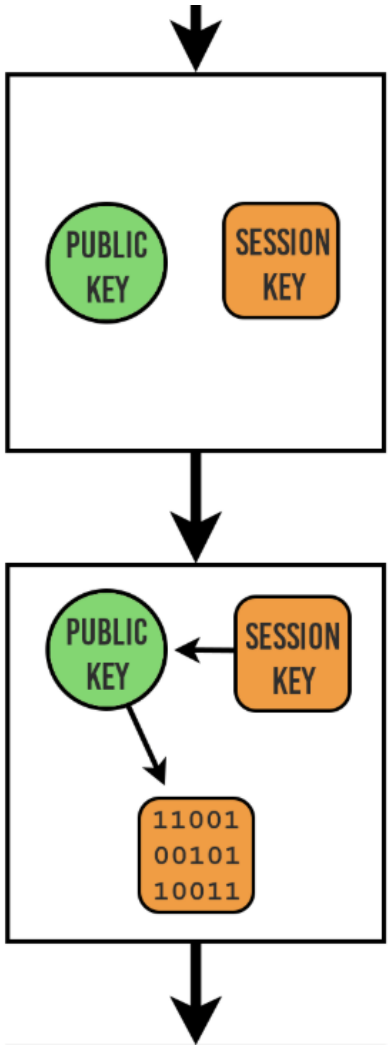
https://tiptopsecurity.com/wp-content/uploads/2017/06/What-Is-HTTPS-Asymmetric.png

Agreeing on a Session Key



https://tiptopsecurity.com/wp-content/uploads/2017/06/How-HTTPS-Works.png

Agreeing on a Session Key

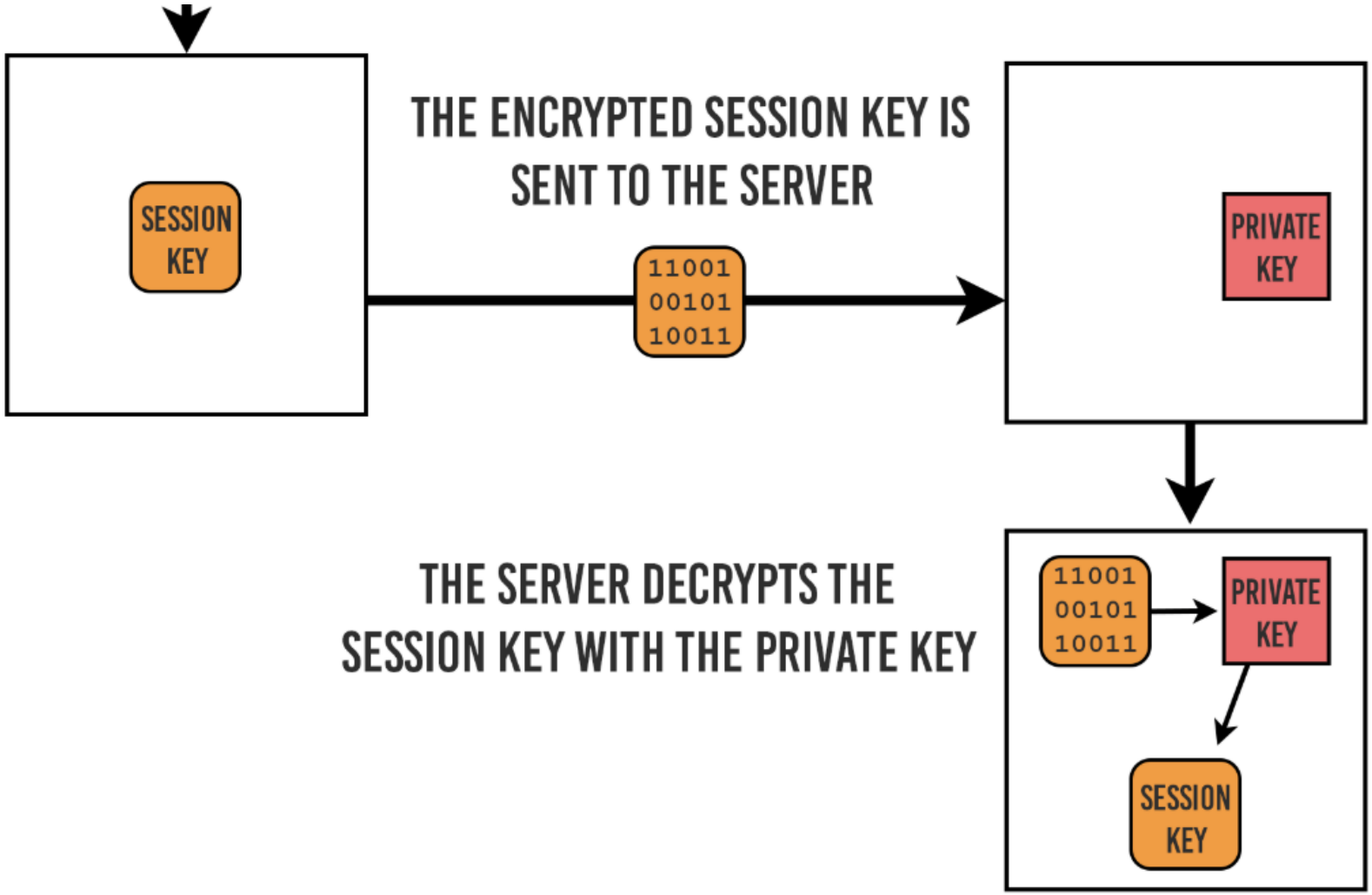


A SESSION KEY IS GENERATED BY YOUR BROWSER

THE SESSION KEY IS ENCRYPTED WITH THE PUBLIC KEY

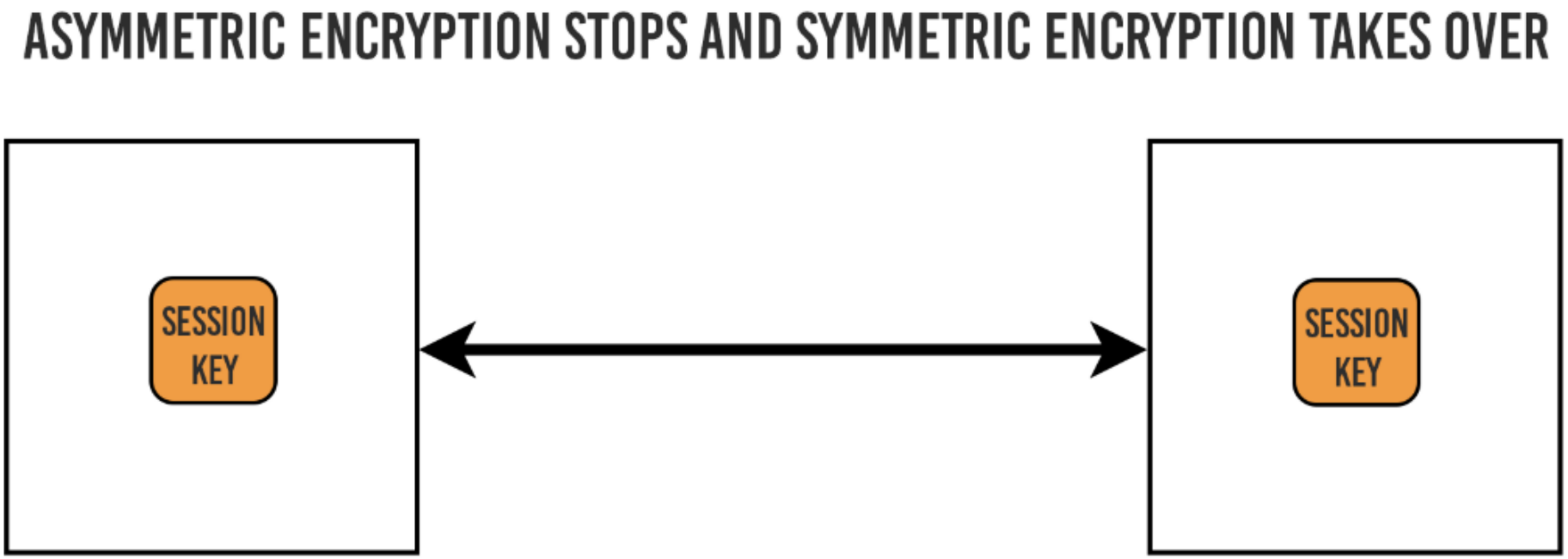
https://tiptopsecurity.com/wp-content/uploads/2017/06/How-HTTPS-Works.png

Agreeing on a Session Key



https://tiptopsecurity.com/wp-content/uploads/2017/06/How-HTTPS-Works.png

Agreeing on a Session Key



This is a big benefit as symmetric encryption is significantly less expensive computationally

https://tiptopsecurity.com/wp-content/uploads/2017/06/How-HTTPS-Works.png

Authenticating the Server

The previous process involves deciding on a shared private key to encrypt/decrypt information exchanged between client and server

After the client initiates the request, the first step is the server sending its public key

But another important question is – how does the client ensure it is talking to the right server?

Authenticating the Server

Without ensuring this, anybody could intercept the original request and respond to the client with their own public key

The client would then think it is talking to server X, when really it is sharing private information with some other server Y

Authenticating the Server

To perform this sort of authentication, we have certificate authorities (CAs)

These CAs are responsible for issuing certificates to the owners of servers

The CAs act as trusted third parties between the client and server

Authenticating the Server

Information with a certificate can include:

The domain name the certificate is issued for

The person/organization it was issued to

Which CA issued the certificate

The CAs digital signature (used for validation)

Issue and expiry dates

The public key of the server

Authenticating the Server

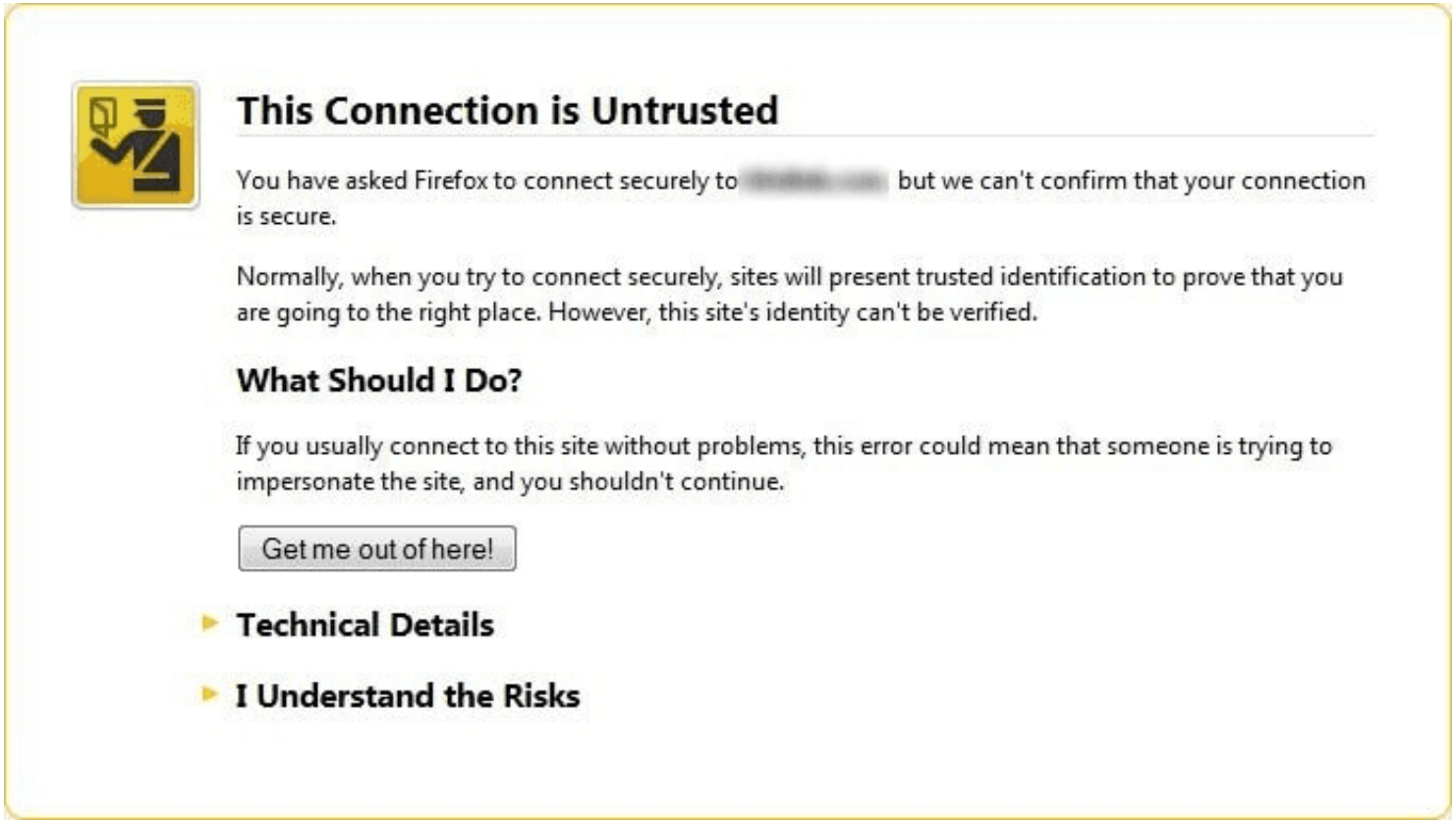
So when a client connects to a server initially, it will receive the server's certificate

The client must decide that it trusts the certificate, usually by deciding if it trusts the CA that issued it

Browsers come with pre-loaded trusted CAs

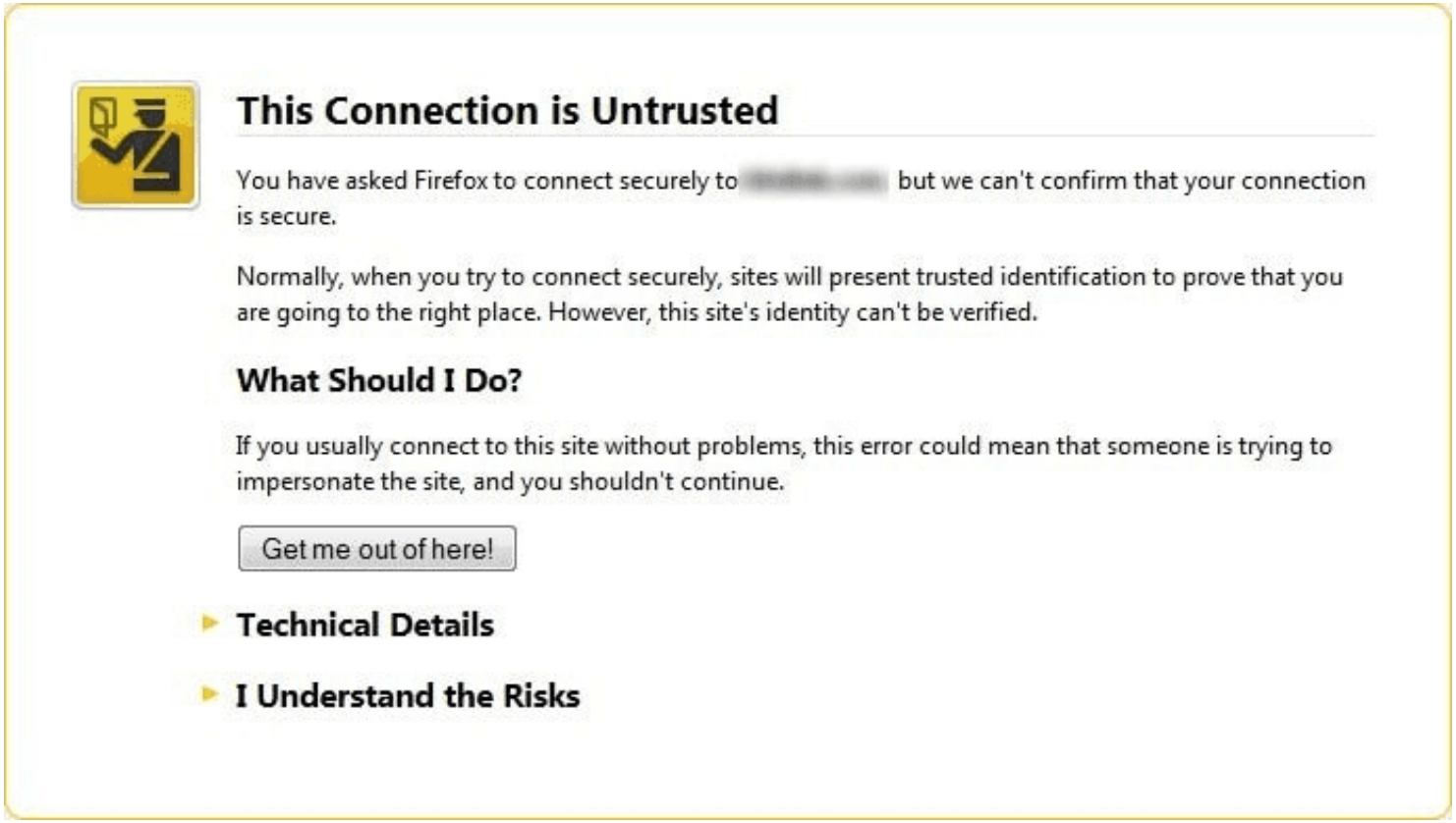
Authenticating the Server

If the browser does not trust the issuing CA, you may see something like this:



Authenticating the Server

You can then decide if you want to trust the site you are connecting to or not…



HTTPS and Node

To get started with HTTPS in Node.js, you can create your own certificate and key

This won't be trusted by browsers, but the user (e.g., you) can decide to trust it anyway

HTTPS and Node

To generate your own certificate and key, you can use the OpenSSL tool

This comes pre-installed on most Unix-based systems

You can also install a Windows version (e.g., from https://slproweb.com/)

To generate your key/certificate:

```
openssl req -nodes -x509 -new
-keyout server.key -out server.cert
```

req: request generating utility
-nodes: do not encrypt private key
-x509: creates an X.509 certificate
-new: generate a new request
-keyout: the key file to write
-out: the certificate file to write

The previous command will create server.key and server.cert files

These contain your private key and certificate

Your public key is contained in the certificate

HTTPS and Node

You can then use the HTTPS module in Node.js and specify your key/certificate files

You can also tie Express into the HTTPS module

See 20-ex1-setting-up-https.js

LetsEncrypt

If you want to open your server to the public, you will want to get a trusted certificate

LetsEncrypt is a non-profit certificate authority

They provide a relatively painless method for getting a trusted certificate/key for your site

'Mixed' HTTP/HTTPS

It is possible to support both HTTP and HTTPS simultaneously:

```
const express = require('express')
const https = require('https')
const http = require('http')
const app = express()

http.createServer(app).listen(80)
https.createServer(options, app).listen(443)
```

'Mixed' HTTP/HTTPS

Recent practice, however, has moved away from supporting mixing of HTTP and HTTPS

It is better to use exclusively HTTPS if your site deals with private data (or if you just don't like snoopers)