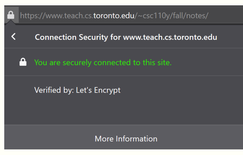
# 8.6 Application: Securing Online Communications

Cryptography is central to all kinds of computing and online communication in today's modern world. Modern security practices inform every stage of how we interact online, from the Wifi networks we connect to, to how data is transmitted back and forth between our computer and a server halfway around the world, and even how data is encrypted for storage on those servers. Every time we visit a website, watch a video on our phone, or post a photo or tweet, we are relying on modern cryptography to keep our communications private.

In this section, we will tie together our study of cryptography by looking at one specific link in the chain of Internet communication. While doing so, we will explore some of the real-world design decisions and trade-offs that go into implementing a secure communication protocol used by billions of people around the world.

## *HTTPS and the Transport Layer Security protocol*

Whether you are browsing a website on your computer or on your phone, you can probably see a little padlock icon next to the website's URL. Here's what happens when you click on it:







This icon is our web browser's way of telling us that the the data being sent from the server (`www.teach.cs.toronto.edu` in our above picture) has been encrypted using a communication protocol called *HTTPS*.[1] This protocol consists of two parts:

- **HTTP (Hypertext Transfer Protocol)**, which governs the format of the data being exchanged between your computer and the server.
- **TLS (Transport Layer Security)**, which governs how the data formatted by HTTP is encrypted during the communication process.

On its own, HTTP allows your computer to communicate with servers around the world. But when combined with TLS, those communications are secure and cannot be "snooped" by an eavesdropper (at least not easily!).

An analogy here might be helpful. Suppose you're living in pre-Internet times, and writing a book (or set of course notes!), and want to send a draft to your publisher through mail. *HTTP* corresponds to the format in which you deliver the book: perhaps chapter by chapter, with a table of contents in front and appendices or an index at the end. *TLS* corresponds to how you encrypt the contents of what you send in this format. For example, you might apply a Caesar cipher to shift every character in your book or you might enclose each chapter in a separate locked briefcase for which only you and your publisher know the combination. Of course, TLS is much more sophisticated than either of the example "security" approaches. For the rest of this section, we'll study how TLS uses the concepts we've learned across this chapter to encrypt your online communications.

[1] We won't define the term "protocol" formally in this course, but you can think of it as an algorithm where the steps are split among two (or more) parties, rather than just a single computer. For example, the Diffie-Hellman key exchange is more commonly referred to as a *protocol* rather than an algorithm.

## *TLS: An overview (simplified)*

For our description of the TLS protocol, we'll use the term *client* to refer to your computer and *server* to refer to the website you are communicating with. TLS starts off with the client initiating a request to the server (e.g., when you type in a URL into your web browser and press "Enter"). The following happens:

1. When the client initiates the request, the server sends a "proof of identity" that the client has connected with the intended server, which the client verifies. *This communication is not encrypted.*
2. Then, the client and server perform the Diffie-Hellman key exchange algorithm to establish a shared secret key.[2] *This communication is not encrypted either.*
3. All remaining communication (e.g., the actual website data!) is encrypted using an agreed-upon symmetric-key cryptosystem, like a stream cipher.

That's it! While the protocol seems straightforward, there are a few real-world details that we'll look at. Let us investigate two questions:

1. Why is symmetric-key encryption (rather than public-key encryption) used to encrypt the communication in step 3?
2. Given that the first two steps of of TLS are unencrypted, how can the client be sure it is actually communicating with the intended server the whole time?

[2] A new secret key is chosen every time you visit a given website. This provides *forward secrecy*, which means that if an attacker records your communication with a server across multiple sessions, but is only able to discover what your key for a single session, they can only decrypt your communication for that session rather than all your past sessions.

## *Why symmetric-key encryption?*

Our first example of symmetric encryption, Caesar's cipher, shows just how old the idea is. Public-key encryption is, relatively, much more modern and does not require that the two communicating parties share a secret key. But modern doesn't always mean better—TLS relies on symmetric-key encryption because public-key cryptosystems, like RSA, are significantly *slower* than their symmetric-key counterparts. While RSA relies on modular exponentiation as the key encryption and decryption steps, modern symmetric-key cryptosystems use faster operations[3] to encrypt and decrypt data.

[3] Typically these operations act on swapping or combining individual bytes in computer memory.

When computers became household commodities, performance was king. Here, performance is a broad term that typically refers to how quickly a computer can do something. For example: how long does it take to encrypt the frame of a video, send it over a wireless connection, and decrypt that frame on your phone? Consider that your phone is likely streaming at least 30 frames per second in order for you to enjoy a video of reasonable quality. It's also increasingly likely that, today, the frame of video is high-definition, which requires even more data to be encrypted and decrypted. While security and privacy is king in today's world, performance cannot be forgotten.

## *Who am I connected to?*

The first two steps of the TLS protocol are "setup" steps for the actual communication of data between the client and server. While a symmetric cryptosystem is used to encrypt the communicated data, these setup steps are unencrypted, and raise a natural question: how do we know we are communicating with the right server?

For example, when we visit `www.google.com`, and our computer performs the TLS protocol with a distant server, how do we know our computer is connecting to a real Google server, and not some fake server that's simply pretending to be Google? The consequences of establishing a connection with such a "fake Google" server are severe: that server might give us manipulated or fake search results, save our login information, or store text, images, and videos we upload to Google Drive or YouTube. Even if we encrypt all of this data in Step 3 of TLS, that encryption does not protect us from a malicious fake server posing as an honest one.

In order to avoid such a dangerous situation, we need some way to verify that the server (e.g., Google) we intended to speak with is actually who they say they are. Herein lies one of the main benefits of public-key cryptosystems. Every public-key cryptosystem, including RSA, can implement two additional algorithms to:

1. Sign message using the private key
2. Verify a signature using the public key

These algorithms allow a user to *sign every message it sends* with is private key, and then have the client *verify* each message signature using the server's public key. We call these **digital signatures**, and they help us identity exactly who we are speaking with. We won't go into the specifics of the algorithms here, but the process for the RSA cryptosystem is similar to what we've outlined in this chapter (i.e., they exploit modular arithmetic). Alice can add her signature, which is a function of her private key, to a message. Bob can verify that Alice is the sender with Alice's public key.

Digital signatures are used in each of the first two steps in the TLS protocol, which is what we'll look at next.

### *Establishing identity: digital certificates*

In the first step of TLS, we said that the server sends the client a "proof of identity". To make that more precise, the data the server sends in this step is called a **digital certificate**, which has identifying information for the server, including its domain (e.g., `www.google.com`), its organization name (e.g., "Google LLC"), and its *public key*.

But how do we know this digital certificate is the "real" one? The certificate also includes the digital signature of a *certificate authority*, which is an organization whose purpose is to issue digital certificates to website domains.[4] So when the client "verifies" the digital certificate provided by the server, what's actually happening is that the client is verifying the digital signature provided by the certificate authority, using the certificate authority's public key.[5]

[4] The largest of these worldwide are IdenTrust and DigiCert, though a recent non-profit called *Let's Encrypt* launched in 2016.

[5] You might ask: how does the client know the certificate authority's public key? It turns out that web browsers come *pre-installed* with the public keys of many certificate authorities!

### *Maintaining identity during Diffie-Hellman*

After Step 1 of TLS, the client is confident that it has connected with the right server. But we aren't in the clear yet—because the Diffie-Hellman algorithm is performed unencrypted, there is still the danger that an attacker might wait for Step 1 to complete and then intercept the messages for Diffie-Hellman in Step 2. Thus the attacker tricking the client into sharing a secret key with the attacker instead of the intended server.

The server's digital certificate doesn't help here! Instead, the server *signs all messages* it sends during the Diffie-Hellman algorithm, so that at every step the client can verify that the message came from the intended server. Of course, this relies on the client knowing the server's public key, which it gets from the digital certificate in the previous step!

It is this *digital signature* from the server that allows the client to consistently verify that it is communicating with the server, and that the messages haven't been tampered with. At the end of Step 2, the client and server have a shared secret key, and can now communicate safely using symmetric-key encryption.

## *(In)effectiveness of Cryptography*

We've mentioned that Diffie-Hellman and RSA are secure because it is very difficult to extract the private part of the data from what is being publicly communicated. But what if it wasn't that difficult? Remember that both RSA and Diffie-Hellman rely on very large prime numbers. But, as we saw in Chapter 7, generating these prime numbers is costly. And it turns out that, unfortunately, many servers use the same group of prime numbers.

Recall that Diffie-Hellman relies on the discrete logarithm problem being difficult to solve. But some steps of the algorithm can be precomputed for a specific group of prime numbers. In 2015, a team of academics discovered that 82% of servers used the same 512-bit group of prime numbers. The team proposed the Logjam attack, which exploited this vulnerability and compromised communications. They also extrapolated that Logjam applied to the 1024-bit case. Today, 2048-bit keys are used to avoid the Logjam attack—for example, Google announced in 2013 that it switched from 1024- to 2048-bit keys.

The Logjam attack is not an isolated incident. Security protocols are constantly being revised, leading to important updates for web browsers, email clients, servers, etc. Earlier versions of the TLS protocol (1.0 and 1.1) are deprecated as of March 2020, which means that "secure" communication must use more recent versions of the protocol. Nor are attacks limited to cryptography. The security and privacy of our data can be attacked at multiple points, and attackers are not limited to exploiting weaknesses when we communicate data. The fields of computer security and data privacy are becoming one of the most important problems to solve as laws and policies slowly catch up to a world where a person's private information is used as a common commodity sold and exchanged by corporations.