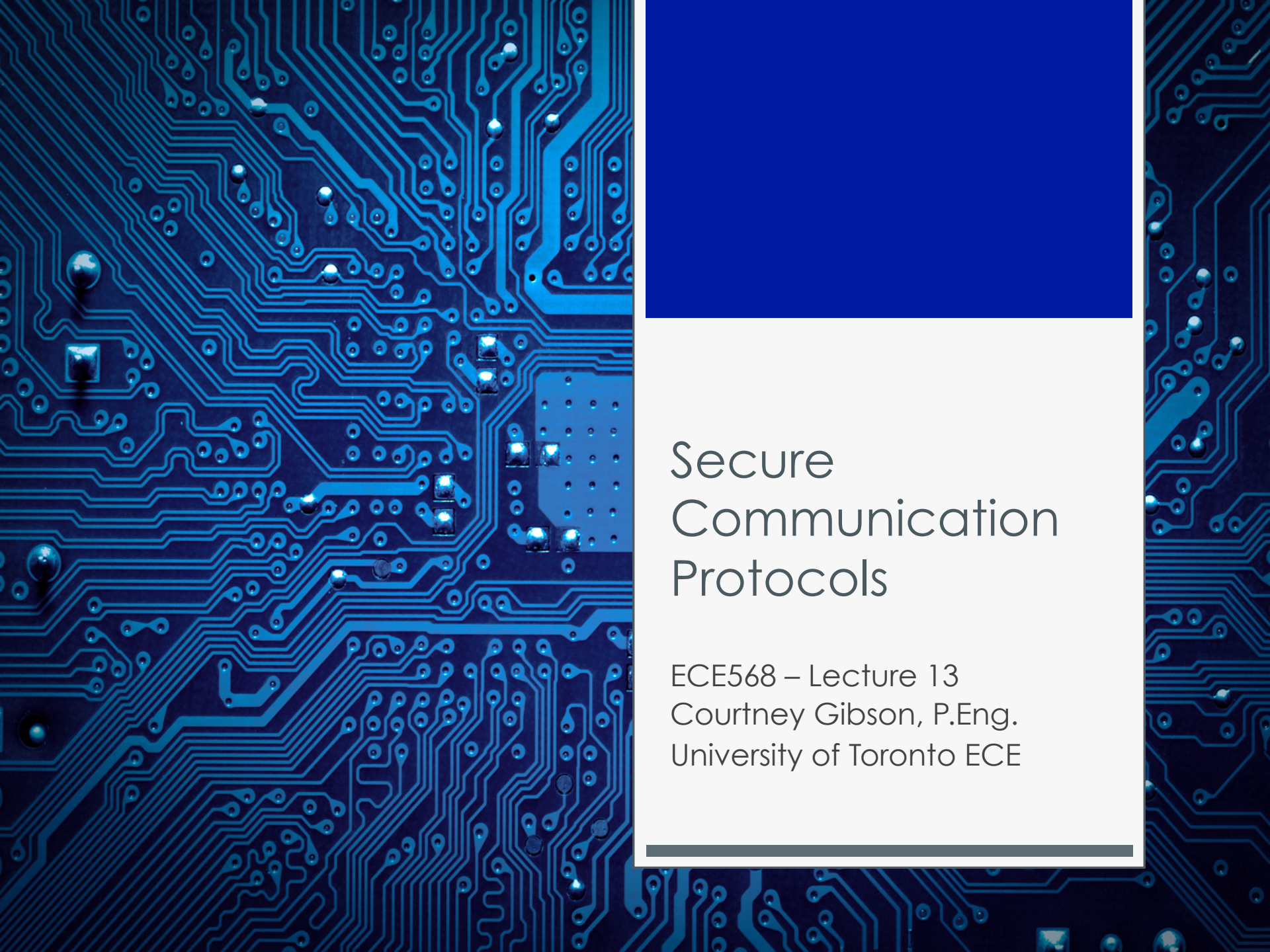


Secure Communication Protocols

ECE568 – Lecture 13

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Communication Protocols

- Attacker goals
- Attack model

Common Protocol Attacks

- Spoofing or message forgery
- Replay attacks
- Re-ordering and splicing attacks

The SSL Protocol

- SSL Handshake
- SSL Communication
- Security of the SSL Protocol
- Performance implications of SSL

Communication Protocols

Communication Protocols

There are several steps involved in designing a communication protocol

- Deciding who will be communicating and what will be sent
- Deciding the guarantees the protocol should provide
  - Confidentiality
  - Integrity
  - Authentication
- Deciding what cryptographic algorithms to use and the modes in which to use them
- Identifying the attacker's goals and abilities

The Attacker's Goals

The attacker could have several goals

- **Key Recovery:** the attacker recovers the key
  - This attack is most damaging
  - Attacker can decrypt all messages
  - Create new fake messages
- **Plain Text Recovery:** the attacker can recover the plaintext content of an encrypted channel
- **Message Forgery:** the attacker can create fake messages and make them look authentic; two types:
  - **Selective Forgery:** The attacker can choose the contents of the forged message
  - **Existential Forgery:** The attacker can forge a message, but can't control its contents

The Attack Model

It is also important to have an accurate model of the attacker's abilities (in increasing power)

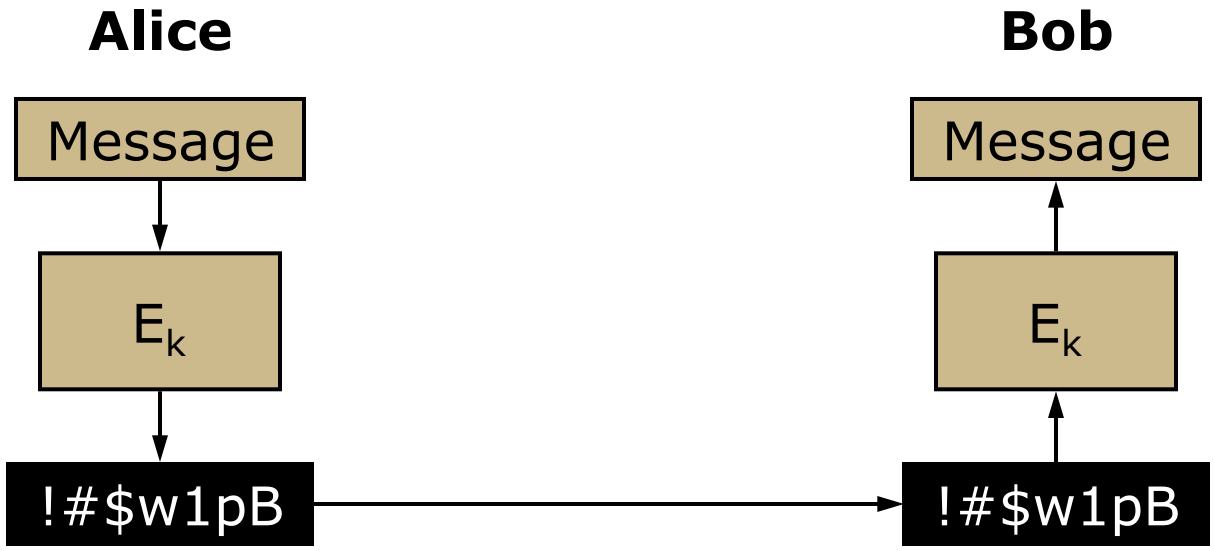
- **Passive Attacks**
  - The attacker can listen to messages (**release contents**)
  - Can record messages for offline analysis (**traffic analysis**)
- **Active Attacks**
  - The attacker can create or modify messages (**spoofing**)
  - Repeat previous messages (**replay**)
  - May be able to prevent communication (**denial of service**)
- **Adaptive Attacks**
  - Attacker learns something with each modified message
  - Uses that to create the next modified message

Common Protocol Attacks

Example Protocol

Alice wants to send something to Bob

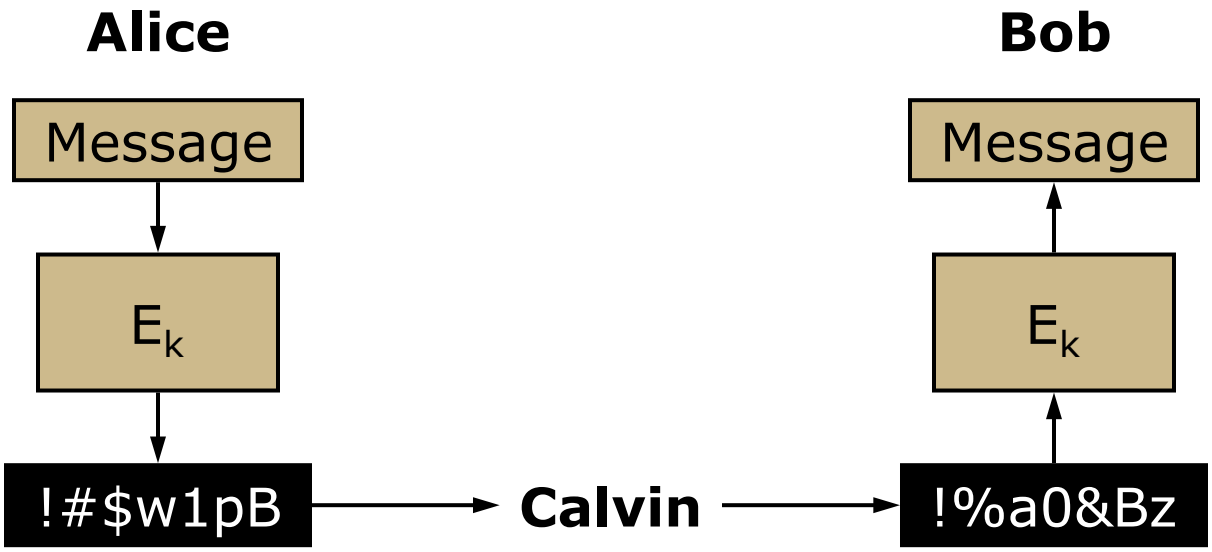
- They securely establish a shared secret key **K** and Alice encrypts the message and sends it to Bob



Spoofing Attack

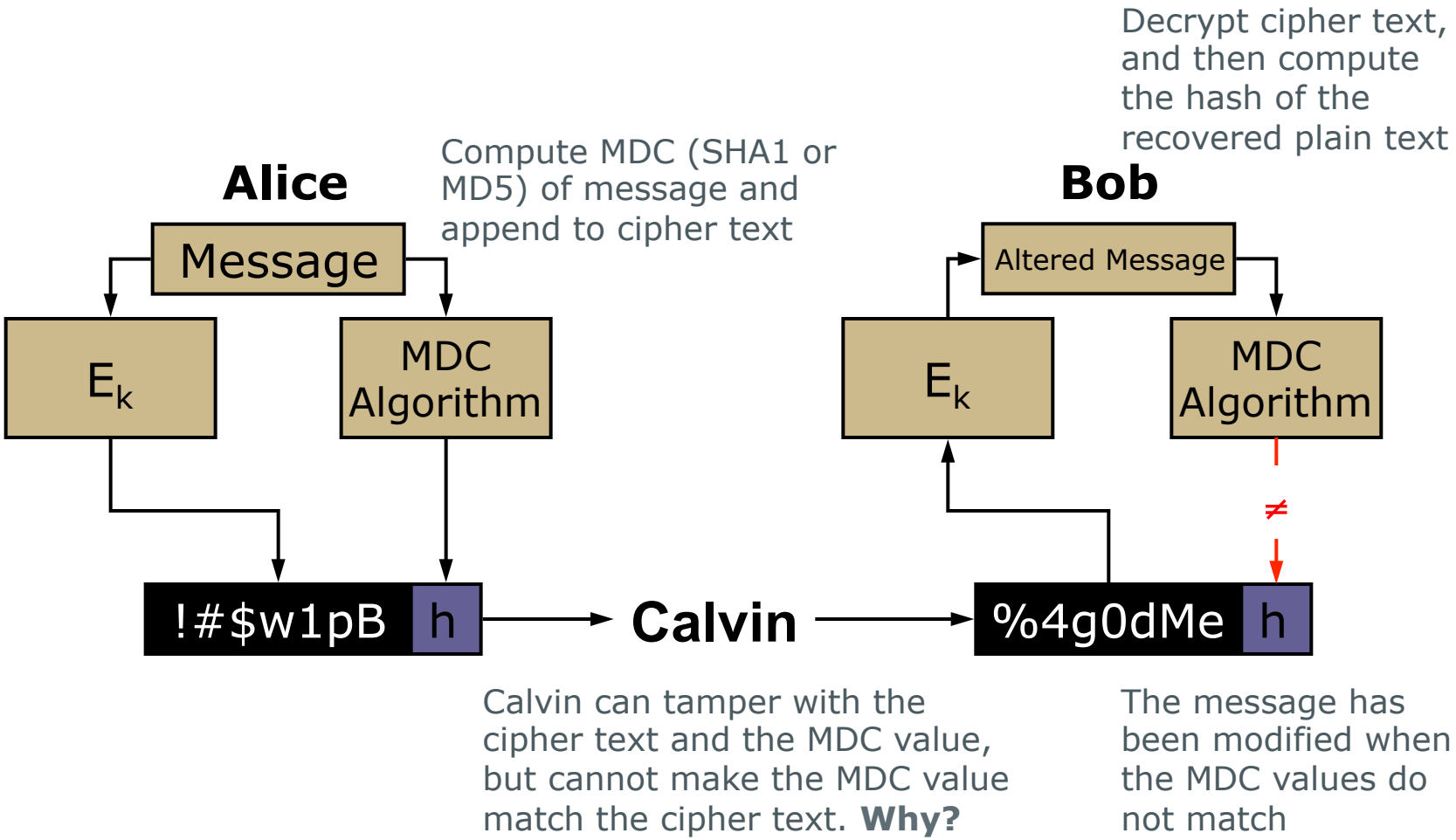
Calvin performs a forgery by substituting some other message for the cipher text

- When Bob decrypts it, he can't tell that the message has been modified



Preventing Spoofing

Adding an MDC prevents the spoofing attack

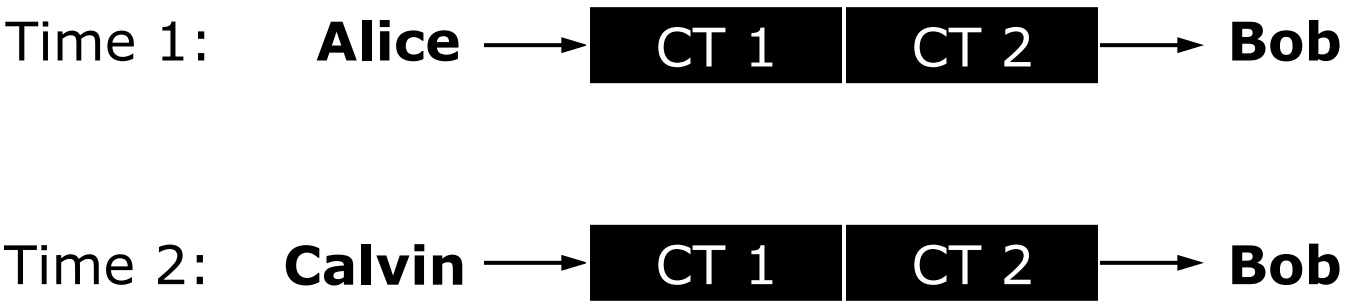


Always take a hash of the plain text, never cipher text!

Replay Attack

Suppose, Calvin can't create valid messages, but he can record messages between Alice and Bob:

- He can then re-send a old message to Bob
- The valid replayed messages will contain valid MDC/ MAC values, so Bob cannot detect that they are not authentic



Preventing Replay Attack

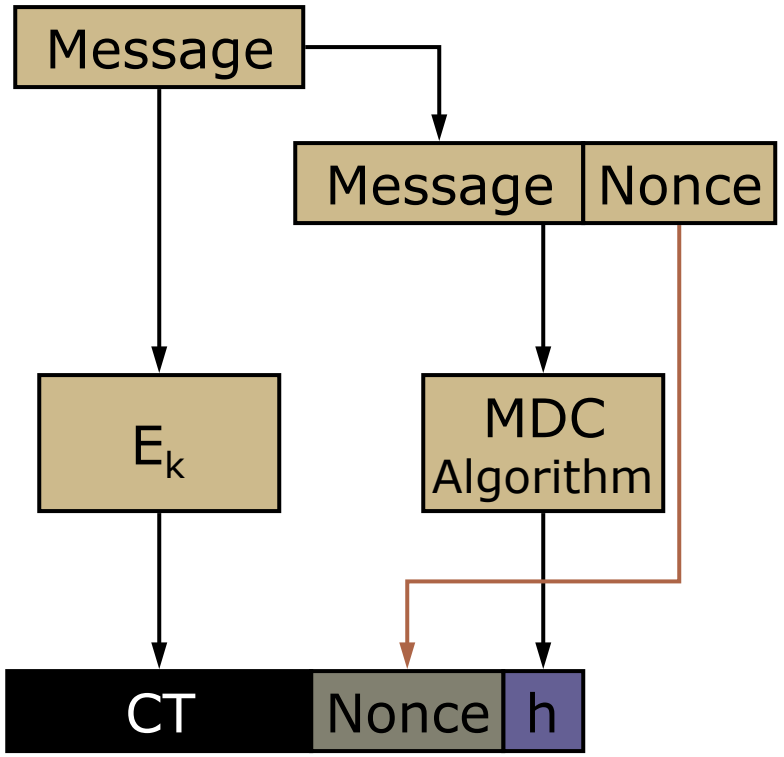
Alice and Bob can prevent a replay attack by appending a unique value to each message

- Once Alice uses a value, she never uses it again
- Bob keeps track of all values he's seen and ignores a message if he sees a value a second time

A unique one-time value is called a **nonce**

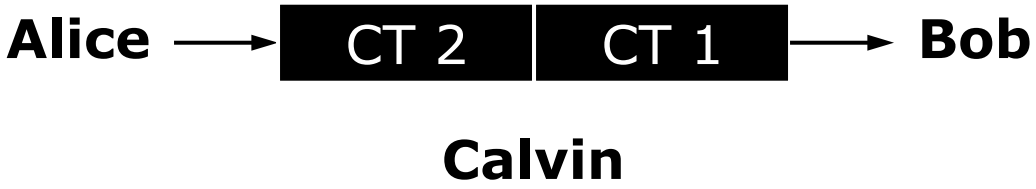
- Hash Value = H(M | | nonce)

Why is it important to include nonce in hash?



Reordering Attack

- Calvin can buffer messages sent by Alice to Bob, and send them to Bob in a different order



- Doesn't CBC protect against reordering?

Preventing Reordering Attack

Alice and Bob can agree to introduce sequence numbers into their messages

- If messages come out of order, or a sequence number is missing, Bob can detect that tampering has occurred
- This also prevents Calvin from dropping messages

- Hash value = H(M || Sequence Number)

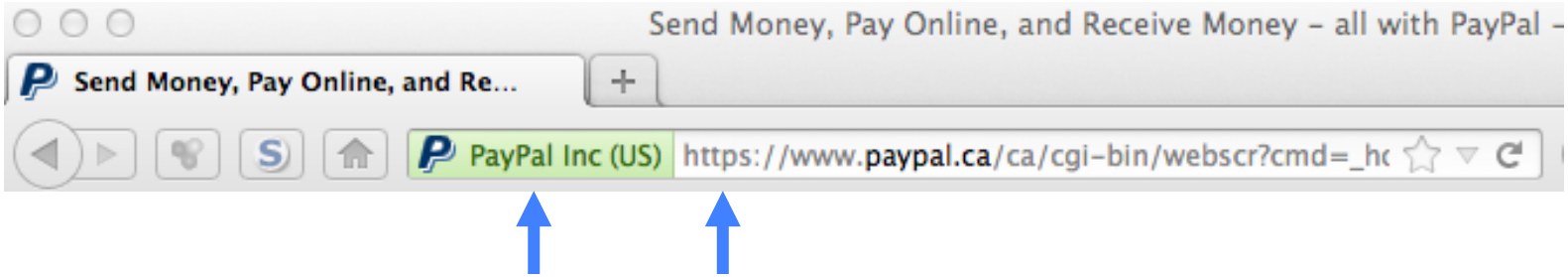
- Why is it important to include sequence number in hash?

The SSL Protocol

The SSL Protocol

The browser is requesting financial information that will be transmitted across the web

- Since packets pass through untrusted routers between the customer's machine and PayPal, the bank information, etc. is encrypted so that any intermediate routers cannot read or alter this information
- The protocol used to encrypt and transfer this information is called **SSL**
- The **https** in the URL and the box in the URL bar (Firefox) indicate that SSL is being used to transfer the information with an authenticated third party



Overview of SSL

The **Secure Sockets Layer** (SSL) protocol was first designed by Netscape in 1996

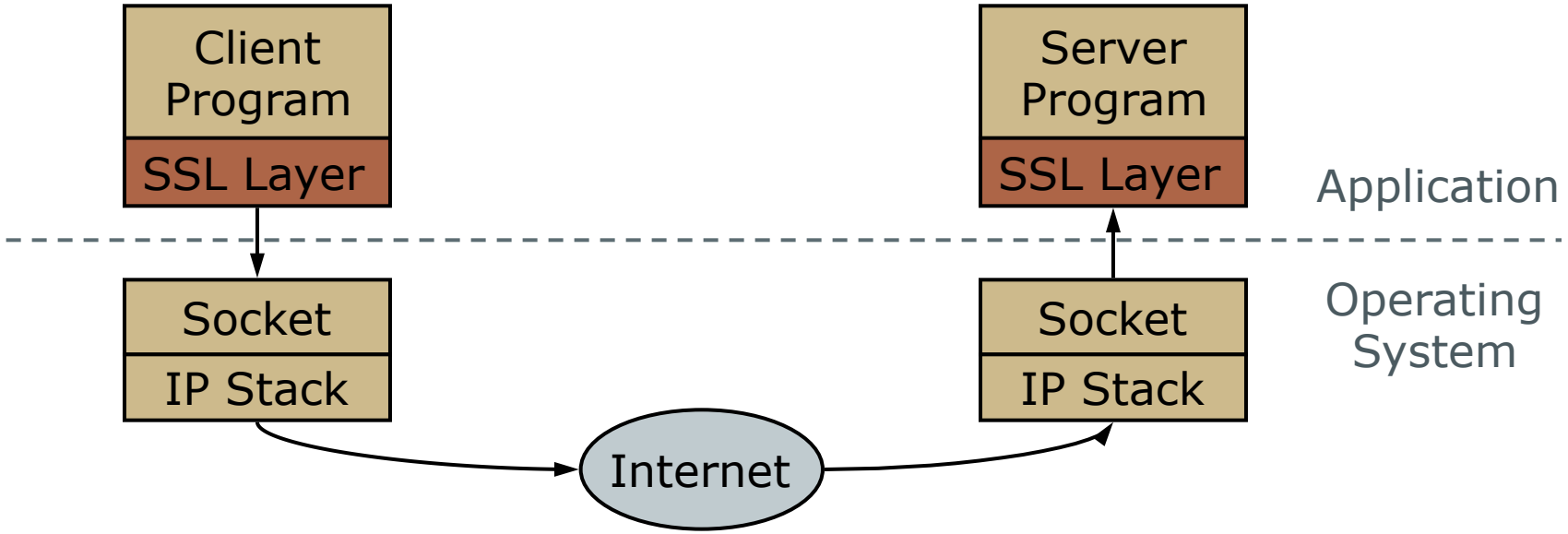
- SSL has undergone several revisions, latest is v3.0
- SSL v3 has been standardized, and the standard is called **Transport Layer Security** (TLS)

- SSL is most commonly used to secure web sessions
  - However, it can be used to secure any application, since it replaces regular sockets
- SSL requires application support at both ends
  - However, the network does not need any SSL support
  - This is often called **end-to-end** security

Using the SSL Protocol

SSL is implemented as a application level library, linked with both the client and the server

- SSL presents the same functionality as a socket, except all information is encrypted before sending it to the socket and decrypted after receiving it



SSL Protocol Phases

SSL has two phases:

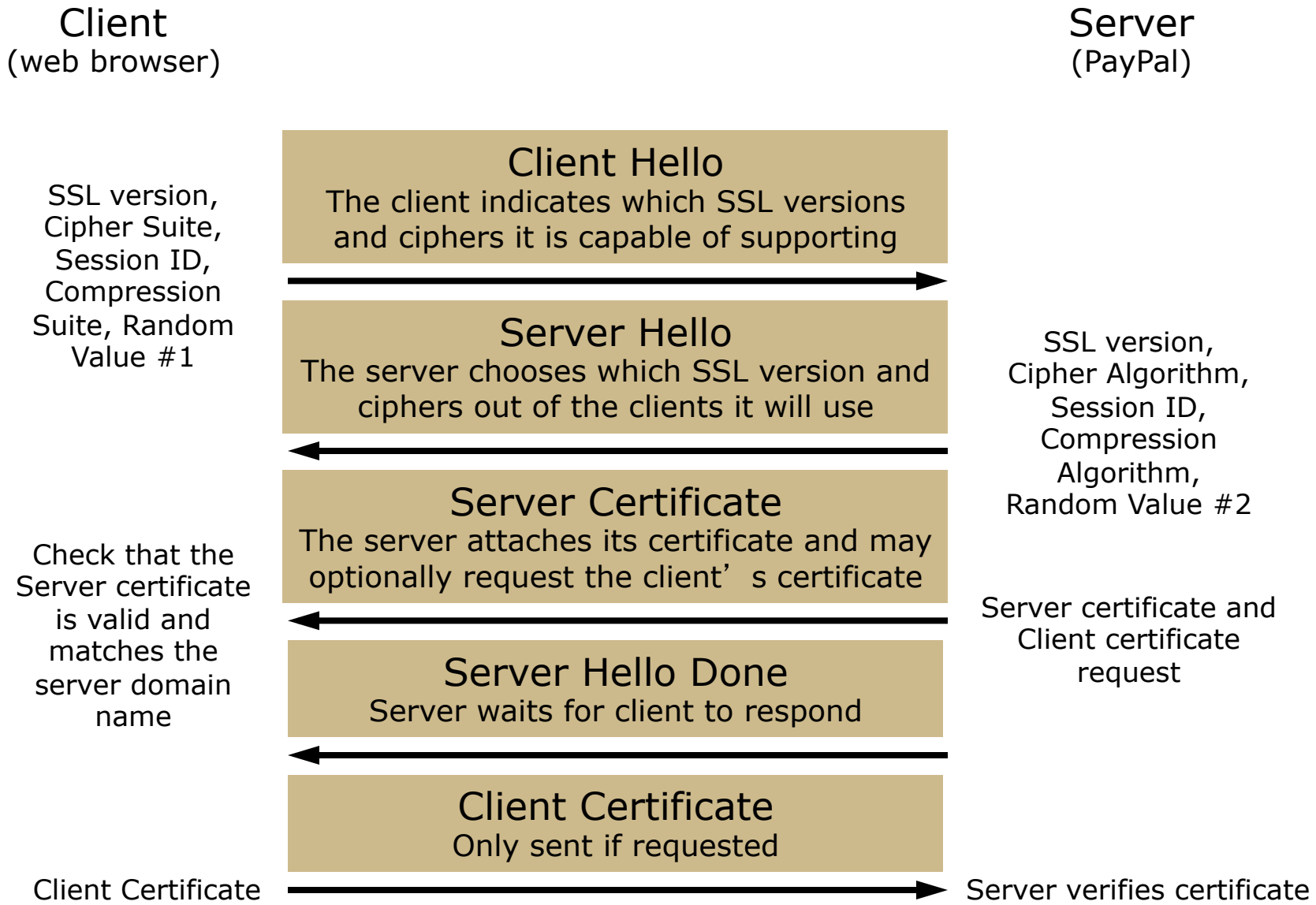
- **Key exchange or handshake**
  - Establishes compatibility between versions, sets up secret key between sender and receiver, and performs authentication
  - Since this happens only once for any exchange, the efficiency of this phase is less critical
- **Communication**
  - Once the keys are setup, an arbitrary number of messages can be exchanged between the two parties in both directions
  - Large amounts of data can be transmitted, so this phase needs to be efficient

SSL Handshake

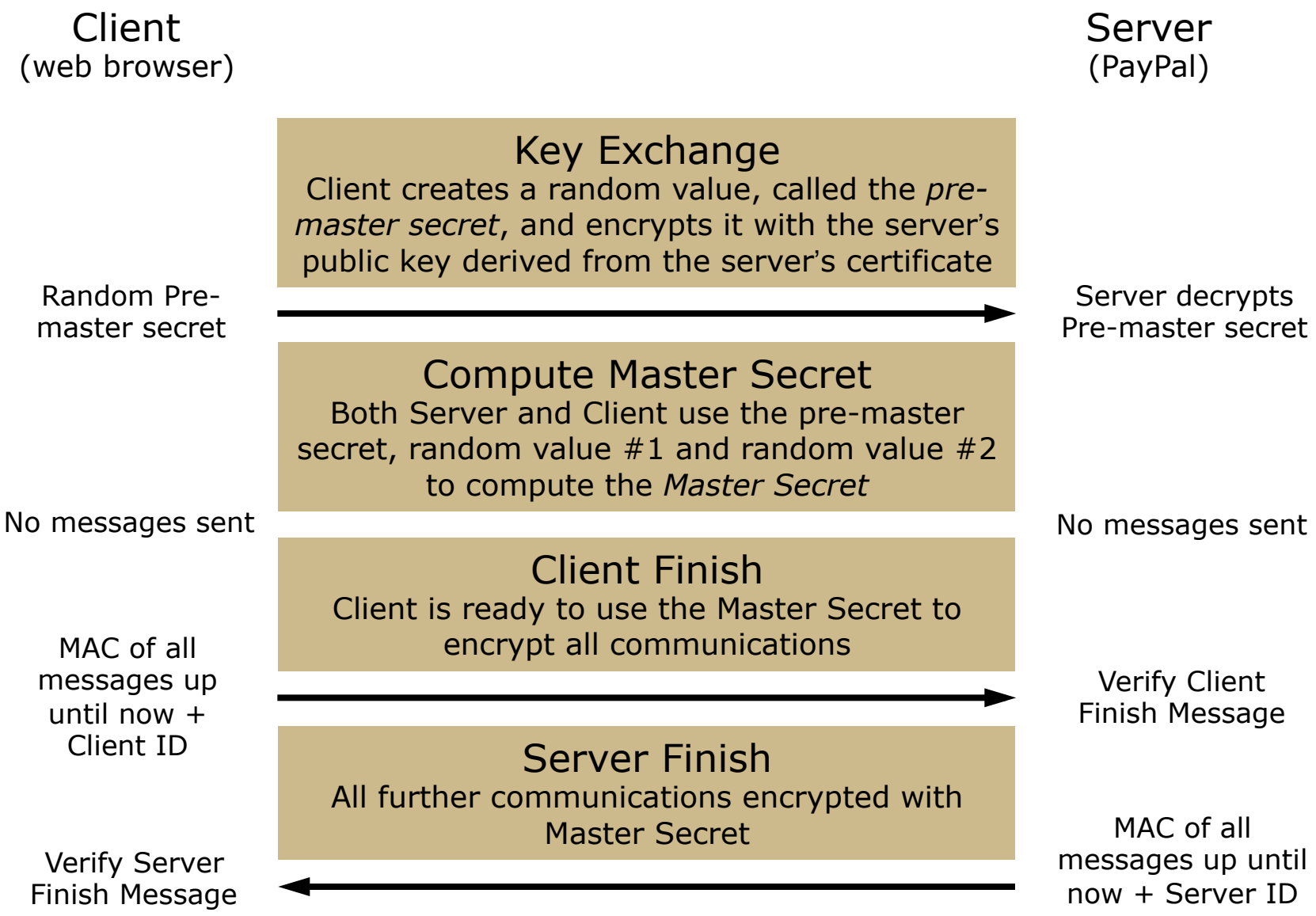
The first phase, SSL handshake, takes place in three steps:

- Establishes the suite of ciphers each side supports, and what version of the protocol is being used
- Securely establishes a shared secret that can be used as a session key for symmetric encryption
- Authenticates each others' identities, via certificates
  - Note this authenticates the identities of the machines, not the users (*i.e.*, people/companies) making the requests
  - User authentication (*i.e.*, a website asking you for a username/password) is not done by the SSL protocol, but by scripts or applications on the web server
  - Note that client machine authentication is optional, and usually not done for every SSL interaction (since web servers will generally connect to any client)

SSL Handshake Detail



SSL Handshake Detail



Handshake Security Features

Protection against spoofing

- MAC at the end of the handshake contains information for all previous handshake messages

Protection against reordering and deletion

- Same as above, think of all the handshake messages for each side as “one” packet

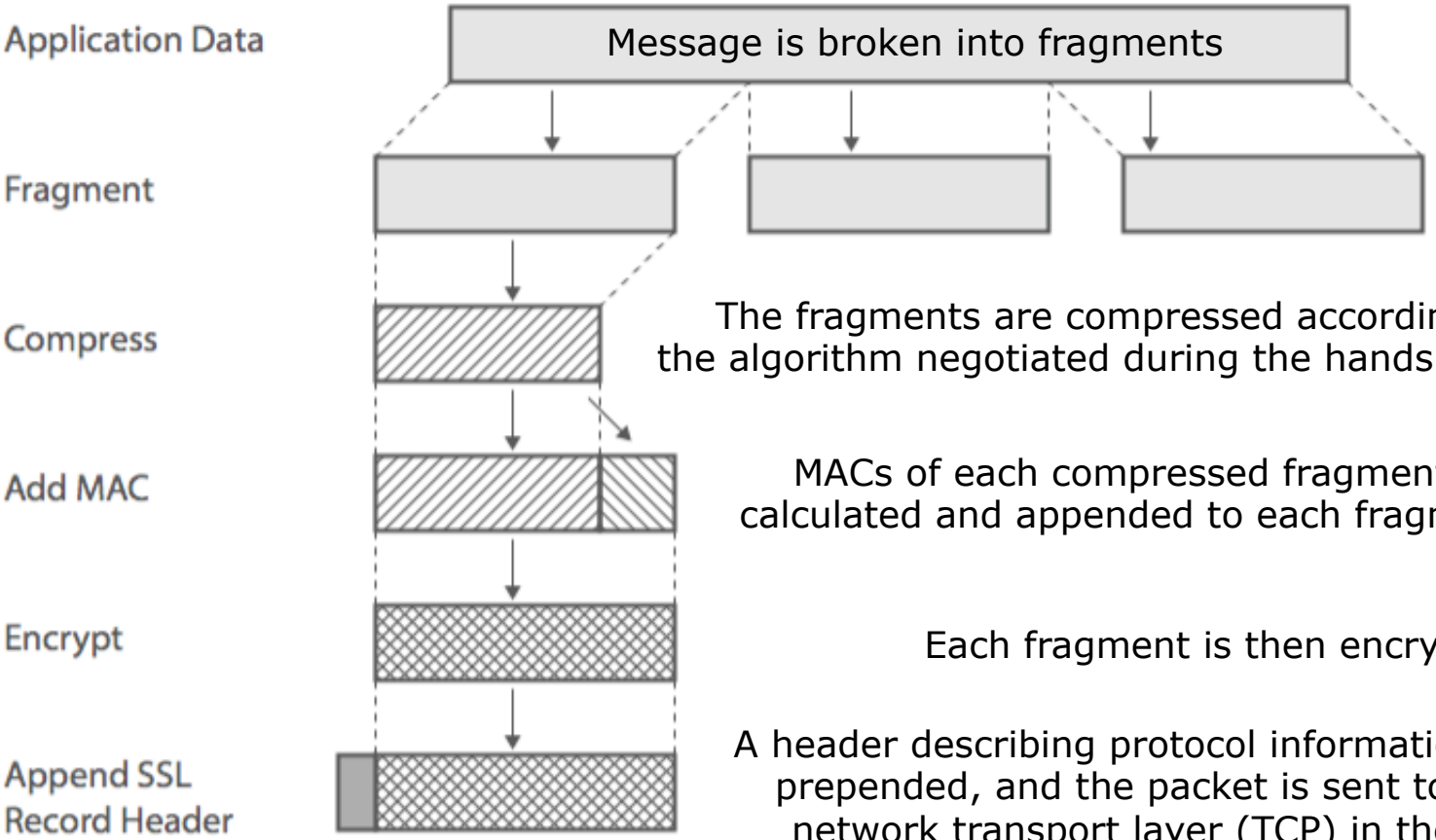
Protection against replay

- Handshake cannot be replayed since nonces are used by both sides

Protection against man-in-the-middle attacks

- Certificates are used to authenticate public keys used for encryption

SSL Communication



The fragments are compressed according to the algorithm negotiated during the handshake

MACs of each compressed fragment are calculated and appended to each fragment

Each fragment is then encrypted

A header describing protocol information is prepended, and the packet is sent to the network transport layer (TCP) in the OS

SSL Communication Security Features

Protection against spoofing

- All encrypted fragments are accompanied by a MAC

Protection against reordering and deletion

- Fragments are numbered using sequence numbers, which are used when generating MAC

Protection against replay

- Fragment sequence numbers are 64-bits long, so they are unlikely to wrap, and thus act as a nonce as well

Performance Issues with SSL

SSL does not impose much performance penalty

- Data is encrypted using a symmetric block cipher (fast)
- Server uses asymmetric decryption during handshake (~1000 times worse than symmetric cipher)
  - For servers that do a lot of SSL communication, handshake becomes the bottleneck
  - To help, the protocol specifies a “Resume” command so if client and server have previously performed a handshake, they can reuse previous security parameters

Hardware acceleration is available (SSL offloading)

- Dedicated hardware can perform public key operations
- Symmetric accelerators also exist for sites that transfer a lot of traffic

OpenSSL

OpenSSL is an open source implementation of the SSL protocol

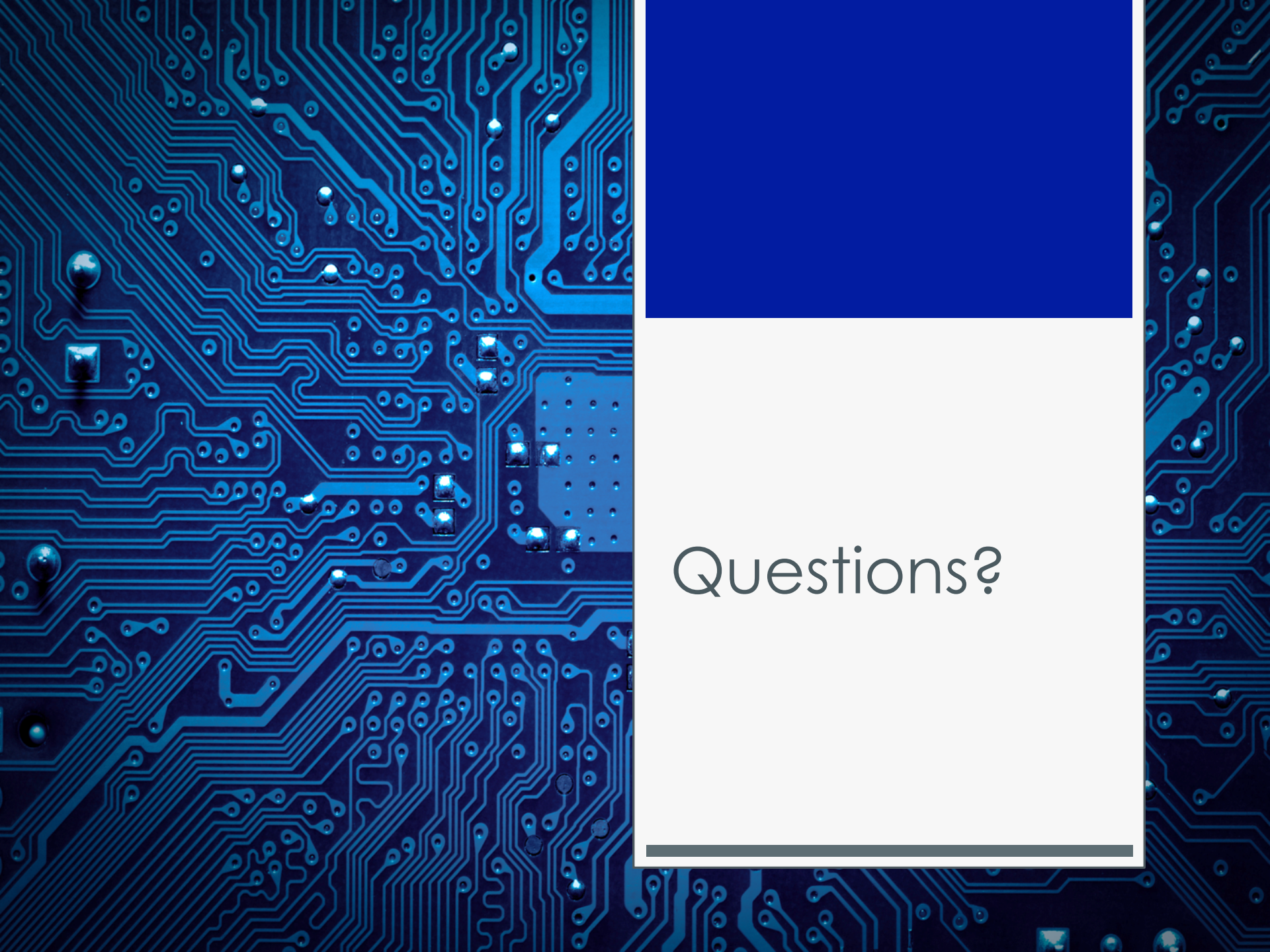
- It's very complete, used in Apache web servers to implement SSL
- It is open source and widely used, but, unfortunately:
  - It's not the easiest to use
  - There is a bit of learning curve
- Some documentation, though not always complete
  - "man" pages, documentation on website:
    - http://www.openssl.org
  - Tutorial information available on course website
  - When in doubt, look at header files
    - **`/usr/include/openssl`**

OpenSSL Primer

- Library needs to have certain parameters initialized
  - **SSL_library_init**
- Create an SSL "context" before connecting to the other side. The context contains settings such as:
  - Versions of the SSL protocol to support for the connection
    - See **SSL_CTX_*** functions
- Create a UNIX socket and bind the SSL context to the socket
  - See **SSL_set_bio** or **SSL_set_fd**
- Perform the SSL handsake
  - Use **SSL_connect** for client or **SSL_accept** for server

OpenSSL Primer

- Check the certificate
  - Can verify the signature on the certificate
  - Check aspects of the certificate (common name, etc…)
  - Use **SSL_get_*** functions
- Communicate over the channel
  - Can use **SSL_write** to write data to the channel and **SSL_read** to read from the channel
- Terminate the channel
  - Shutdown the channel with **SSL_shutdown** to notify the remote side that you're terminating the connection
  - It's good practice to notify the other side, or else a **truncation attack** is possible, in which the adversary forces one side to drop the connection, and the other side thinks the transmission is over


Questions?