Lecture Notes on Feb/13/2023

Chapter 4:
The Secure Sockets Layer

CYENG 351: Embedded Secure Networking

Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)

Chapter 4 Overview

- The **Secure Sockets Layer (SSL)** is the de facto standard for secure Internet transactions.
- It has achieved this status by being not only secure, but being highly generic and imminently practical as well.
- SSL exists in the network layer between **Transmission Control Protocol (TCP)** and your application, providing blanket security to all data transferred over the network.
- The **Application Programming Interface (API)** for an SSL implementation is typically very similar to the standard network sockets API (POSIX-style).
- For this reason, it is simple to transform any plain TCP/**Internet Protocol (IP)** application into a secure Internet application with very little effort.
- The standard is very flexible in algorithm choice and features, so it is easy to pick and choose the features from a requirement list.
- Most PC-based implementations of SSL are **monolithic** - all features are compiled into every application.
- **SSL Format of Design and Usage → Modular and Structural Approach + Optimizations + Hardware Assistance (using FPGA/ASIC)**.
- The advantage of hardware implementations is the ability to use larger keys and achieve the same or better performance.

SSL History

- The Secure Sockets layer had a modest beginning as a project at Netscape Inc. in the mid-1990s to secure Web transactions for their (then-dominant) Netscape Navigator web browser.
- **SSL Version 1** was an internal project that was never released.
- **SSL Version 2** was publicly released and quickly became a standard way of conducting secure web transactions, eventually beating out other secure offerings from other companies such as Microsoft.
- Unfortunately, some poor decisions in Netscape's implementation of SSL version 2 led a couple grad students to successfully break the encryption utilizing attacks on the random number seeds chosen (which included **the system clock value - a number that is easily predicted and controlled**).
- The attack revealed that an SSL-secured connection could be broken by a clever attacker in a manner of minutes.
- As a result of the compromise, SSL version 2 was deemed insecure and unfit for any real transactions.
- Despite this enormous shortcoming, SSL version 2 is still included in many popular web browsers, though it is now often turned off by default.
- You should never enable it, unless you want to try your own hand at breaking it.
- In the aftermath of the SSL version 2 compromise, Netscape scrambled to provide a fixed implementation of SSL.

SSL History (Cont.)

- At the time **SSL Version 3** was started, the **Internet Engineering Task Force (IETF)** began a project to implement a standard version of SSL that was designed to become the eventual standard.
- The standard, called **Transport Layer Security**, or **TLS** (a name generally disliked by the committee members but not hated as much as the other contenders), is now complete and described in **RFC 2246**.
- The Netscape version was introduced to the industry much faster than TLS, and SSL Version 3 soon became the dominant secure web protocol.
- TLS is structurally identical to SSL Version 3, with a few minor enhancements to the way cryptography is handled, so it is sometimes referred to as **SSL version 3.1**.
- SSL version 3 to this day has not been compromised (at least as far as anyone can tell), but it is generally assumed that TLS is more secure and should be used for all new development.
- TLS is beginning to become the dominant standard, but SSL version 3 is still widely used and is still considered a practical way to secure web transactions.
- Due to the close similarity to TLS, SSL version 3 can be supported in a TLS implementation with a minimum of effort.
- We refer to TLS and SSL version 3 collectively as "SSL (Current Version)".

Pesky PKI

- The **SSL protocol** provides blanket security for network communications by utilizing the advantages of both public-key cryptography and symmetric-key cryptography.
- There are definite benefits to public-key algorithms because we don't have to figure out how to safely exchange keys with the person (or system) with which we wish to communicate.
- We can just send out our public-key and in receiving keys from others, we can communicate freely and securely.
- Well, that's not entirely true, since someone could potentially provide a public-key and lie about who they really are in order to entice you into communicating sensitive information to them.
- This exact problem is at the heart of how SSL is typically deployed in web browsers and servers.
- Keep in mind that the way SSL is currently used does not mean it is the only method of authentication that can be used - in fact, the mechanism by which SSL certificates are distributed has nothing to do with the SSL protocol.
- SSL is most frequently paired with a distribution mechanism.

Pesky PKI (Cont.)

- To solve the issue of authentication (namely, **the secure distribution of SSL certificates**), a few companies put their reputations on the line to provide the public with signing services.
- The whole concept of **Public-Key Infrastructures (or PKI)** relies on the inherent trust in companies like Verisign to do a satisfactory amount of due-diligence on each certificate they sign.
- SSL makes PKI possible through the construction of the digital certificates it uses for authentication.
- Since signing a digital certificate with a private RSA key, for example, is able to be chained any number of times back to a common root, one company can provide that root certificate as part of standard web application implementations and extend their trust to numerous third parties.
- SSL is designed to use any number of root certificates for authentication, so just a few companies provide root certificates, and everyone can use the system.
- It is quite possible that a public SSL client would need as much as a megabyte of space to store all the relevant certificates to assure the highest probability of compatibility with any unknown server.
- However, it is likely that most embedded applications will either be an SSL server implementation, or they will not need to connect to every server ever created. → Not well suitable for embedded systems.

PKI Alternatives

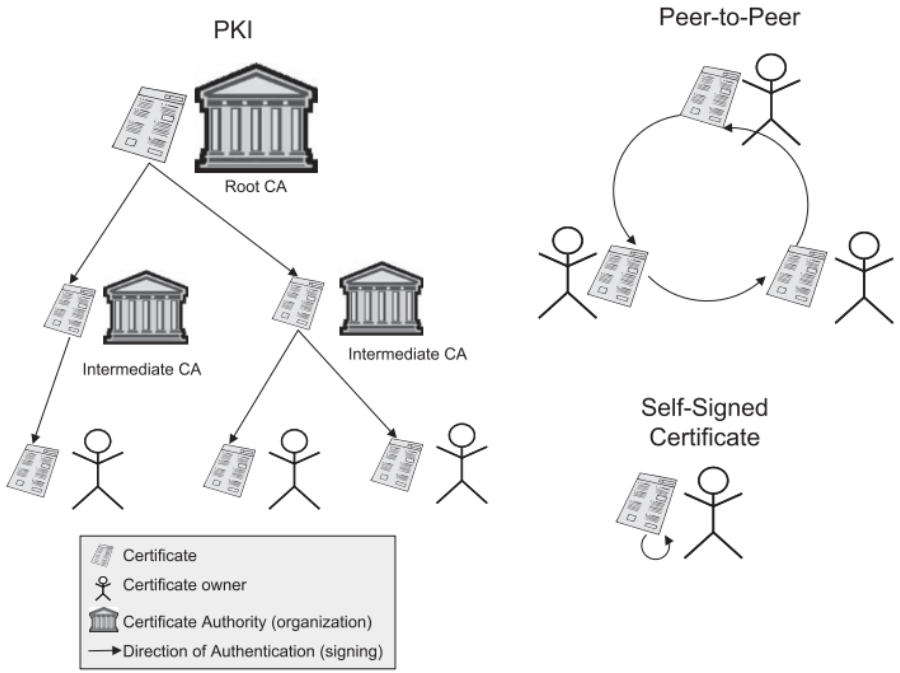
- The **PKI system** used for e-commerce on the **Web** is not the only way to deploy **SSL certificates**.
- Many embedded applications would be hard-pressed to provide the space required to store all the root certificates.
- **PKI** is simply one implementation that has been pushed by large companies for web browsers, but we have the power to do things differently.
- In order to provide authentication for SSL in embedded applications, there are different options for deploying certificates that in many cases are more applicable and practical than PKI.
- The simplest path to SSL authentication is to create **your own self-signed certificate**.
- This essentially means that you will create a certificate with **the appropriate public-key/private key pair**, put the public-key in the certificate, and sign it using the private one.
- As it turns out, you don't have to use Verisign after all!
- Once you deploy your application, **authentication** is simply a matter of using the private key you created to authenticate the certificate being provided by your application.
- If your application is a web server, then you can install the certificate into the browser's root certificate cache - from that point forward, the browser will **always accept that certificate** (assuming it hasn't been tampered with or corrupted).
- Alternatively, if your application is an SSL client, then your server can provide a certificate signed using the same private key, which the application can then verify directly using the single certificate.
- For simple applications, a self-signed certificate is the only way to go. → So, we can create **a self-signed certificate**.

PKI Alternatives (Cont.)

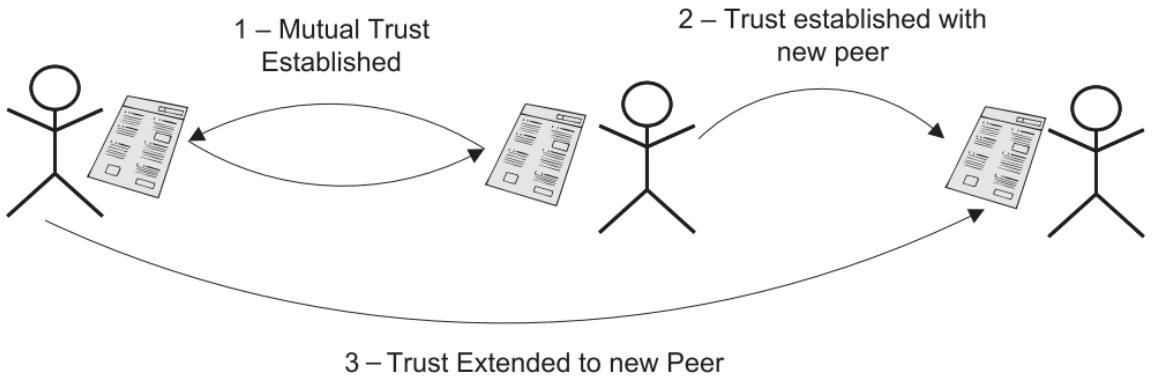
- Remember that we can self-sign a certificate, so there is nothing keeping us from signing other certificates.
- All you need to do is to create **a single root certificate** (which simply means it is **self-signed** and will be used to sign other certificates), and then use that certificate's private key counterpart to sign all your other certificates.
- If you install your single root certificate in a web browser, then you have automatic authentication for all the certificates you create and sign using that certificate.
- This particular method of certificate distribution is quite useful in venues other than embedded systems, such as providing authentication on a corporate intranet, for example.
- **Creating a self-signed certificate and going into the business of being your own CA are both very reasonable ways to provide basic SSL authentication for your application**.
- However, it may be that neither those methods nor PKI are suitable for what you are looking for.
- Fortunately, there are some other ways of distributing trust in the form of certificates, and most of them center on some concept of peer-to-peer networking.
- There are various implementations of peer-to-peer authentication, and the idea hasn't caught on for e-commerce, so we will just stick to the basic ideas behind peer-to-peer authentication mechanisms. → The basic idea is that of extending trust.

PKI Alternatives (Cont.)

PKI Alternatives



Peer-to-peer PKI



SSL Under the Hood

- One of the issues with SSL is keeping **Backward-Compatibility** with older versions. → How is **Forward-Compatibility**?
- It would be better to communicate using **SSL Version 2** than nothing at all (well, not much better, if your information had any value, but you get the point).
- The other action that can occur is that two SSL implementations may actually both speak newer versions of SSL or TLS, and they can negotiate up to the most recent version they both support.
- This backward-compatibility feature can be optional, allowing for newer implementations to communicate directly but the feature is required at the very beginning of the SSL handshake.
- The **SSL handshake** begins with a network connection of some variety, in most cases the connection is a simple TCP/IP network connection (the lower-level and physical layers do not matter).
- SSL is essentially a TCP/IP application, and for the most part does not require any knowledge of the underlying network mechanism.
- That being said, SSL does rely on the network protocol being used to handle dropped packets, so a reliable protocol like TCP will work just fi ne for the lower level, but an unreliable protocol like UDP will not.
- This requirement of SSL does not reduce the security of the protocol or cause any other problems, even if the TCP connection is compromised.
- SSL will be able to detect any attempt to corrupt the information using the message digest authentication mechanism, and the encryption prevents eavesdropping.
- The problem with using an unreliable protocol under SSL is that SSL will flag every dropped packet as a potential attack.
- Once the network connection has been established, the SSL handshake itself can begin.

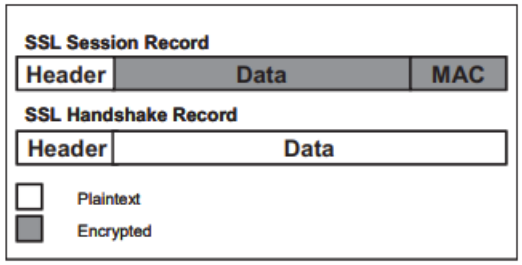
SSL Under the Hood (Cont.)

- The SSL communication layer is similar to other network protocols in that it consists of a header containing protocol information and a body that contains the information being transmitted.
- In SSL, these units are called records and can be thought of as being analogous to the TCP frame and IP packet.
- **In the handshake, the SSL records are unencrypted, since the encrypted connection, called the SSL session, has not yet been established**.
- The *backward-compatibility handshake* begins with an unencrypted SSL record called the Client Hello, which is sent by the SSL client wishing to establish the SSL session with a particular server.
- By convention, the application initializing an SSL connection is always the client, and follows a specific set of actions that correspond to actions on the SSL server.
- A cipher suite is simply a collection of algorithms and some specifications for the algorithms represented by an enumeration.
- For SSL, the cipher suites consist of a public key algorithm, such as RSA, a symmetric-key algorithm such as AES (also called the bulk cipher), and a hashing algorithm.
- The server selects the cipher suite it wants to use based upon its capabilities and the priorities set by the application designer or user for selecting cryptographic algorithms.
- This is the negotiation phase of the handshake—the ciphers actually used for the communication are determined by a combination of what the client offers and what the server can support.
- The server stores away the information from the client (such as the random data used later for seeding the session key generation) and generates a Server Hello message, which is sent back to the client.
- The SSL certificate contains authentication information about the server that the client can use to verify the authenticity of the server, as well as additional information about the server and, most importantly, the certificate contains the public-key that will be used to exchange the information used to generate the session keys used by the secure SSL tunnel once the handshake is complete and the session is established.
- The certificate also contains a digital signature (an encrypted hash) that the client can use to verify the information in the certificate.

SSL Under the Hood (Cont.)

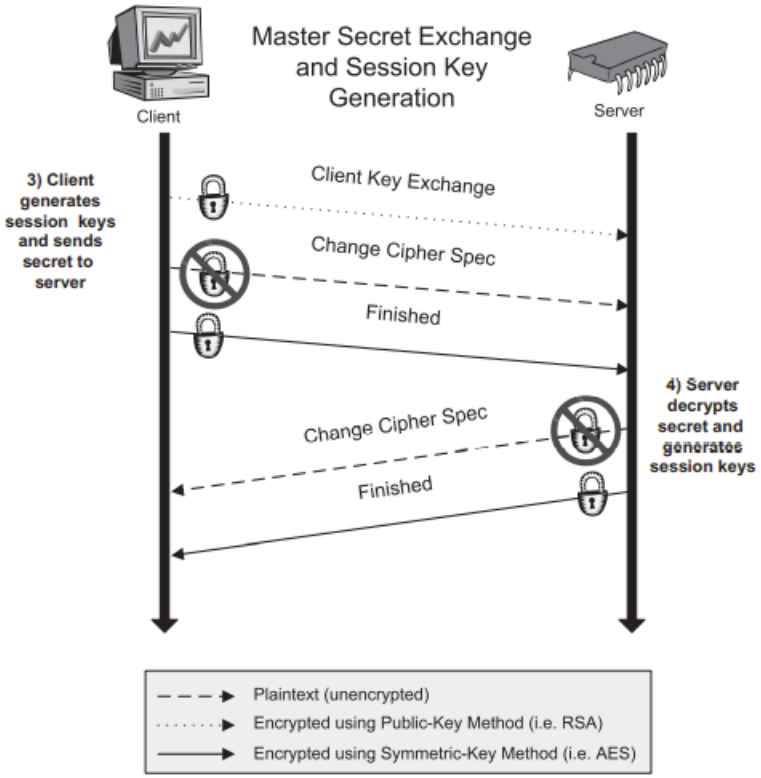
- The digital signature is usually **signed by a third-party Certificate Authority (CA) using the reverse-RSA operation (encrypting using the private key, decrypting using the public-key)**.
- The client stores a number of digital certificates from different CA entities, and uses the public-keys stored therein to decrypt incoming server certificate signatures.
- In this manner, a client can verify an unknown server through the concept of distributed trust.
- If **the client trusts the CA**, and the CA trusts the server, then the client can trust the server as well.
- As long as the CA is reputable and checks up on all the servers it signs certificates for, this system works pretty well.
- SSL is designed so that the server can send any number of certificates to the client, the idea being that the client should be able to use one of the certificates to authenticate the server.
- For this reason, the server Certificate message can be duplicated any number of times, once for each certificate the server sends to the client.
- The **Master Secret** is essentially a cryptographically secure hash of the other data that is always 48 bytes in length.
- The Master Secret is used to generate what is called the key material block, essentially an arbitrary-length expansion of the Master Secret from which the session keys are derived.
- Generation of the key material block is done using a hash expansion algorithm that differs slightly between **SSL version 3 and TLS** (we will not cover what SSL version 2 does).
- **In SSL version 3**, the hash expansion is defined as part of the protocol. **TLS uses a method called P_hash, which is composed of 2 hash algorithms, MD5 and SHA-1, and a hash wrapper** called HMAC.
- HMAC, described in RFC 2104, provides a mechanism to add keying to hash functions.
- Essentially, both the SSL version 3 and TLS mechanisms are the same - **the SSL version 3 mechanism is based on an early draft of HMAC**.

SSL Under the Hood (Cont.)

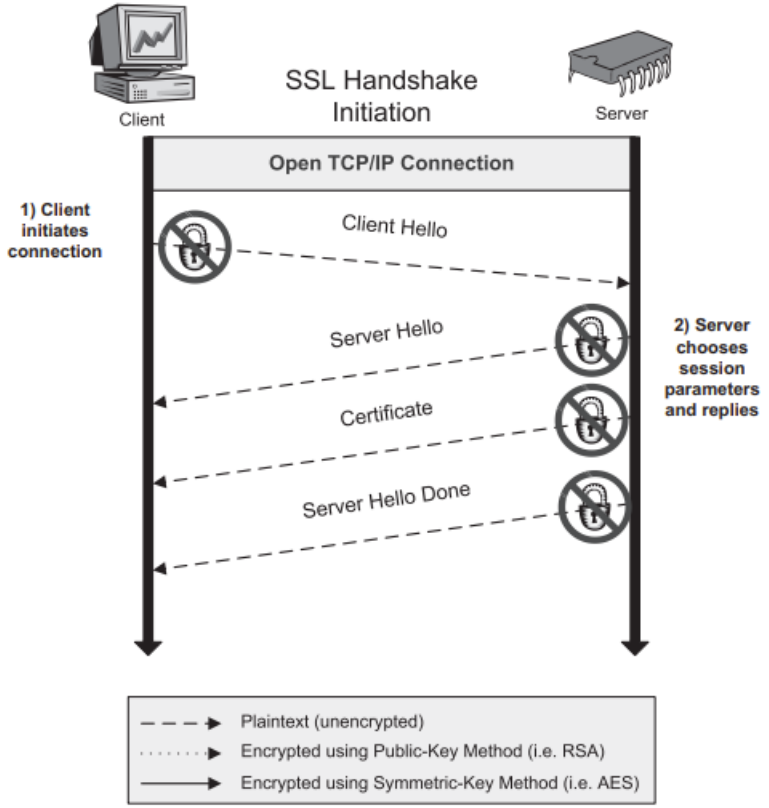


SSL Record

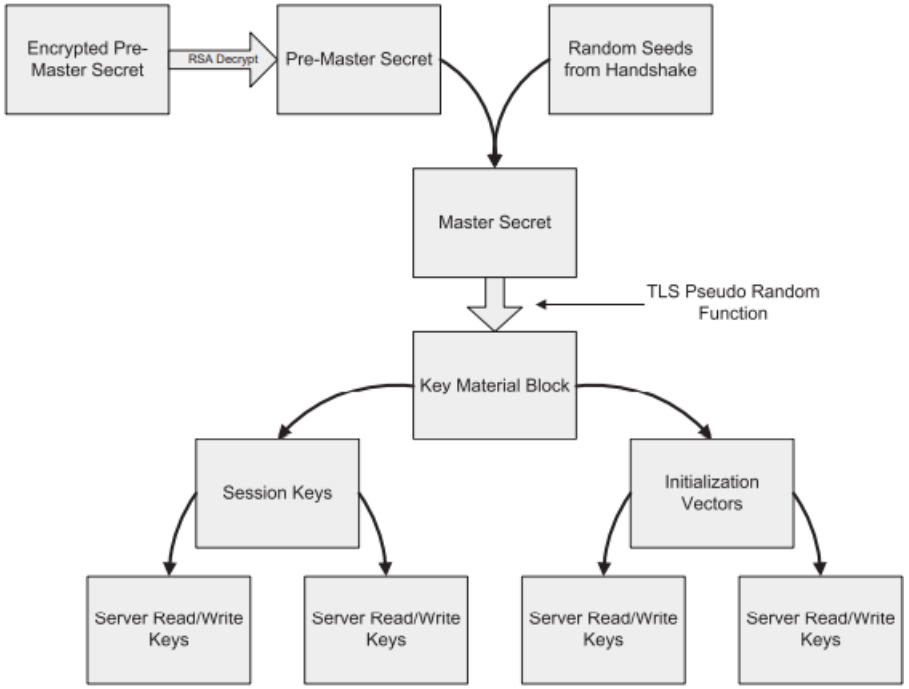
SSL Handshake 2



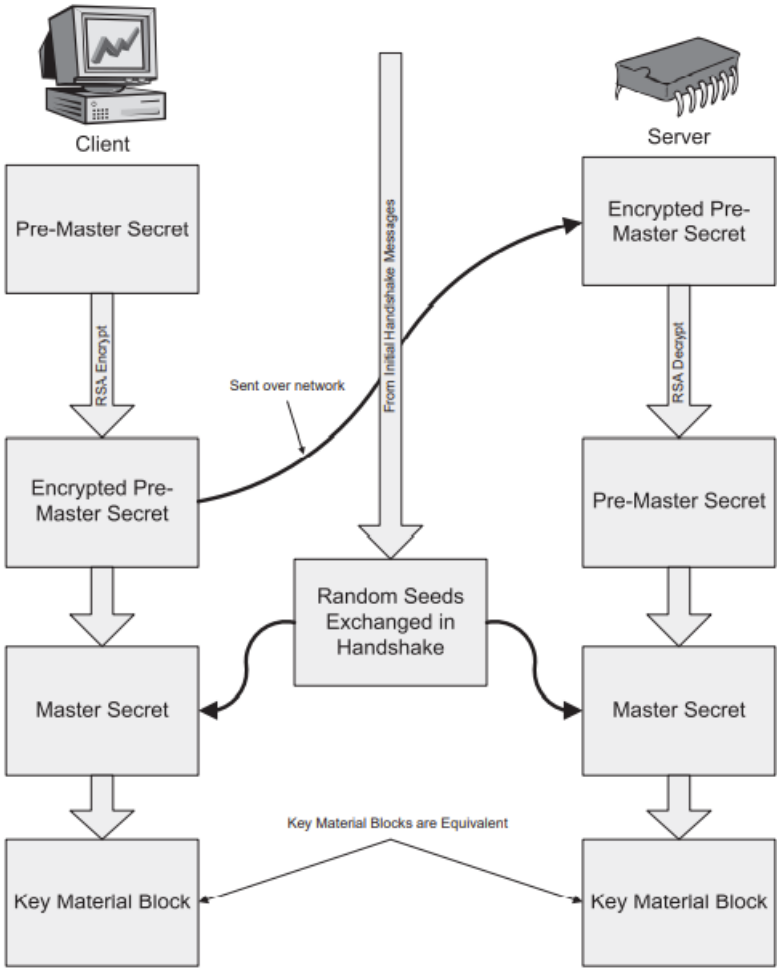
SSL Handshake Part 1



SSL Under the Hood (Cont.)



Key Material Block from Master Secret



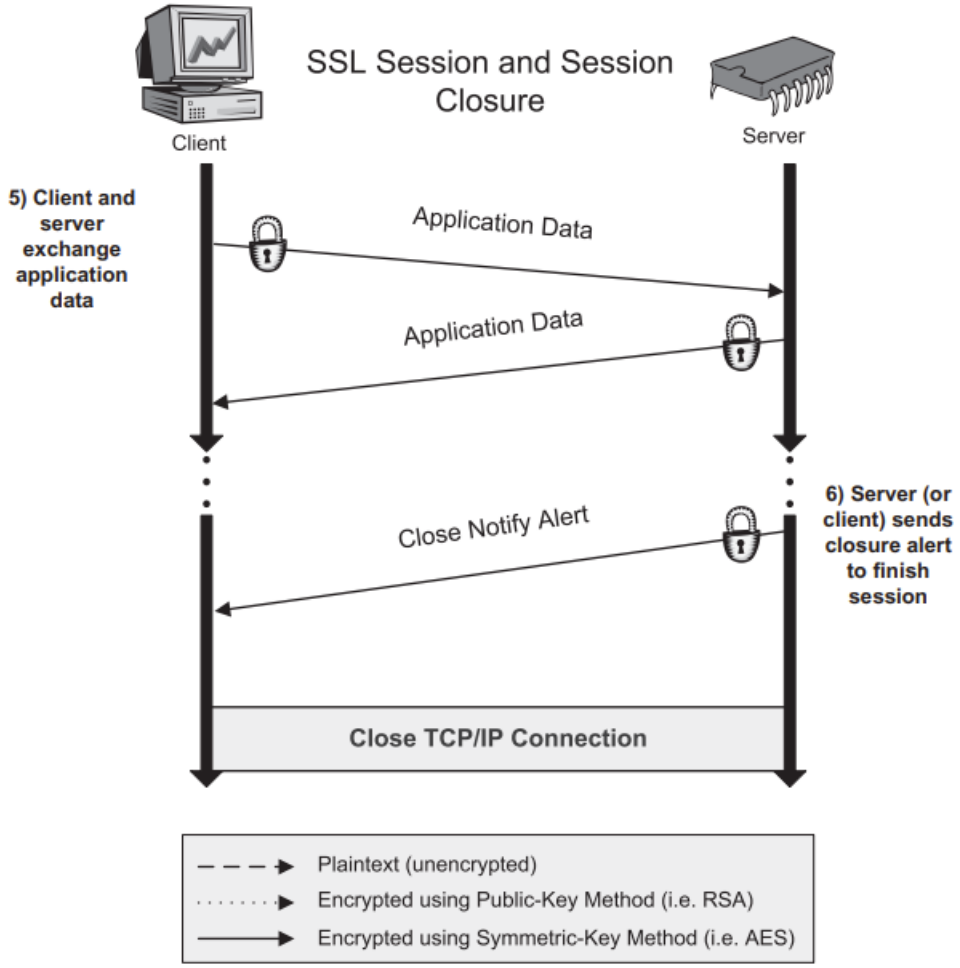
Simultaneous Generation of Keys on Client and Server

The SSL Session

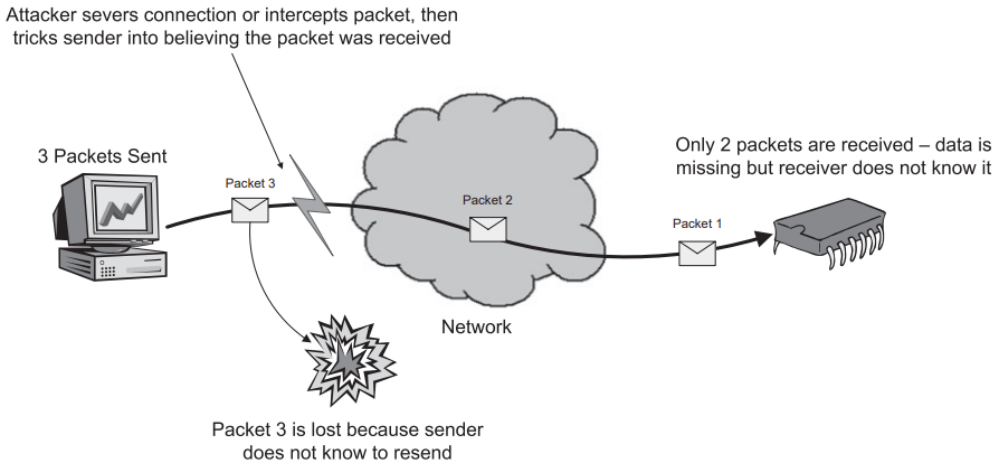
- The handshake makes up the bulk of **the SSL protocol**, since it is the basis for all the security of the session.
- The session itself is fairly boring, since all that really happens is an encryption using the session keys derived during the handshake.
- One difference between a "**regular**" encrypted message and an SSL message (the SSL Record) is the use of a **Message Authentication Code, or MAC**, for verifying the integrity of messages.
- The MAC is a simple hash using an algorithm determined by **the ciphersuite** being used - a couple examples of hashes used are SHA-1 and MD5.
- The **SSL MAC** is actually slightly more than a hash of the data being sent, since it is actually seeded using the MAC secrets derived along with the session keys.
- The purpose of the MAC is simply to verify that the data in the SSL record was not compromised or tampered with, both things an attacker could do to disrupt communications without needing to break the actual encryption.
- The session continues as long as both sides are willing to talk.
- At the end of the session, either the client or the server sends an alert to the other side to confirm that it is finished communicating.
- An **SSL alert** is simply an SSL record containing an encrypted message (again using the session keys) that usually indicates something abnormal has happened.
- In the case of **the final alert**, called **close_notify**, it is actually expected behavior and in fact it serves an important purpose.

The SSL Session (Cont.)

SSL Handshake 3



Truncation Attack



SSL in Practice

- **SSL** is a fantastically successful protocol. → Implemented in practically every web browser and web server, it provides the security backbone for the majority of Internet transactions.
- As the latest incarnation of SSL, TLS is being used even in other protocols as a fundamental building block for security in wireless and mobile applications.
- One reason for the success of SSL is the fact that it has an answer for just about every attack out there and provides multiple layers of redundant security, making it a robust and provably safe protocol.
- Another reason SSL is so widely implemented is that the protocol is flexible - it defines how cryptography should work together to provide security without actually defining what "**cryptography**" should be.
- Even the key generation, which uses standard hash algorithms, can be changed if needed.
- This flexibility is what makes SSL a prime candidate for implementation as a general security protocol for practically any embedded system.
- **The PIC application** does not use SSL directly, but the command protocol that is developed for use with the AES algorithm derives some ideas from SSL, most notably the use of checksums and random data to ensure that each encrypted payload is different.
- **The Rabbit case-study** uses a commercial implementation of SSL to illustrate the use of the protocol with HTTPS and web browsers.

Assignment

- **Reading Assignment:**
  - Stapko, T., 2011. **Practical embedded security: building secure resource-constrained systems**. Elsevier.
    - ✓"Chapter 4: The Secure Sockets Layer", Pages 67-82.

Questions?