Global Information Assurance Certification Paper

Exploring SSL and Its Related Components

Aurora Bataclan

GSEC Practical v.1.4.b (Option 1)
November 2003

Abstract

In a rapidly advancing technological society such as ours, the threat to our privacy grows stronger everyday. It is possible for unethical third parties to spy on our electronic conversations, to steal from us financially by taking our credit card information, or worse yet to use our information to steal our identity. Cryptography (vis-à-vis SSL) allows us to protect our privacy in the face of such threats by enforcing the four cornerstones of secure communication: confidentiality, data integrity, authentication and non-repudiation.

In this paper we will discuss SSL and how it helps us to overcome the security challenges that face us today. We will start off by examining the types of encryption utilized by SSL, followed by a more detailed look at how public-key cryptography works. We will then discuss where SSL resides on the TCP/IP protocol stack, as well as how the SSL Handshake takes place. Finally, we will take a look at certificates, a player in SSL transactions, so that we can gain a better understanding of how clients authenticate server certificates.

Challenges of Online Communication Today

Everyone uses the Internet nowadays, and it has evolved such that many people just can't get along without it. However, due to the nature of the Internet, it is possible for someone to eavesdrop on our online conversations; to tamper with our information as it is being transmitted through the wire; it is possible for an unethical third party to pose as someone else, someone we trust, perhaps an online store or a bank that we rely on for services. When our personal information is on the line, we want to have some degree of confidence that it is protected, that no one can snoop our traffic (eavesdropping). You want to have some degree of certainty that when making online purchases, you truly are giving your credit card information to Amazon or TheGap and not to some hacker posing as your favorite online store (impersonation or spoofing). You want to ensure that the $20.00 you agreed to pay for that sweater doesn't get morphed in-transit to a $2,000.00 charge on your credit card (tampering of data).

Secure Sockets Layer (SSL) is a protocol developed by Netscape whose goal is to secure data transmitted over the Internet. SSL protects us from those hazards mentioned above. It accomplishes this by utilizing a combination of techniques to enforce the four cornerstones of secure communication; namely, confidentiality, integrity, authentication and non-repudiation. Confidentiality is achieved through the encryption of transmitted data; data integrity is achieved through the application of message authentication codes (MACs); authentication,

and non-repudiation are achieved through the use of RSA's Public Key Cryptography. Although SSL is application independent and can be transparently utilized by any number of applications, it is optimized for HTTP. Most people will encounter SSL whenever they go online to shop and pay for their items with their credit card, or when they do online banking. When you visit a secure site and see the "https://" string prepended to your URL (as well as the closed padlock on the bottom right corner of your browser), you are participating in a transaction utilizing the SSL protocol. This means that data exchanged between you and the server you are communicating with is encrypted, a validation of the server's identity has taken place, and you can be rest assured that the data you send to the server will not be altered along the way (or that if it is altered, it will invalidate your transaction).

For purposes of this discussion we will focus on the most prevalent use for SSL – the communication between a web server ("SSL server") and a browser ("SSL client"). It should be noted, though, that this is only for ease of discussion. In actuality, the potential uses for SSL are quite varied and not limited to the HTTP protocol.

The Four Strongholds of Secure Communication

Before we go any further, let us go over the definitions of the four pillars of secure communication: confidentiality, data integrity, authentication and non-repudiation.

a. Confidentiality. This means that when two parties are communicating, it is not possible for someone to eavesdrop on their conversation. Encrypting an online conversation helps to ensure confidentiality because a hacker won't be able to make heads or tails out of the data that is flowing through the wire. There are two kinds of encryption schemes to choose from: symmetric (shared-key) encryption and asymmetric (public-key) encryption. An example of a public-key algorithm would be RSA (developed by Rivest, Shamir and Adleman), and an example of a symmetric algorithm would be DES (Data Encryption Standard, used by the U.S. Government) or Triple-DES (DES applied three times).[1]

b. Data Integrity. This means that a message is not tampered with or dropped while in transit to its final destination. SSL utilizes something called Message Authentication Code ("MAC") to ensure that a message has not been tampered with. MACs are hashing algorithms, a function which you put a message through. Hashing algorithms accept messages of any length, and the output is called a 'message digest', which is of fixed length. Two of the most popular hashing algorithms are SHA-1 and MD5.

[1] Introduction to SSL, http://docs.sun.com/source/816-6156-10/contents.htm

c. Authentication. This means ensuring that the entity you are communicating with really is whom you think it is. This is accomplished by way of public-key encryption. In addition, certificates provide a way of binding a public key to a name. There is, however, a caveat: these methods are effective to the extent that the private key is kept a secret. If one is careless with storage of the private key, and it falls into the wrong hands, then it is possible for the unscrupulous third party to impersonate the true owner of the private key.

d. Non-repudiation. Authentication and non-repudiation go hand-in-hand. If you can prove that a message originated from a certain entity, then that entity cannot later deny having sent the message. That is non-repudiation.

Encryption Types Used by SSL: Symmetric vs. Asymmetric

There are two types of encryption that SSL uses: symmetric and asymmetric. Symmetric encryption means that a single key is used for both encryption and decryption of data:

```
Encryption:  plaintext --> symmetric key --> ciphertext
Decryption:  ciphertext --> symmetric key --> plaintext
```

Figure 1. Encryption and decryption with shared (symmetric) keys.

In the figure above, note that the term plaintext stands for non-encrypted data, while the term ciphertext represents encrypted, unreadable data. This type of encryption is very fast, providing high network throughput. The downside to symmetric key encryption is that it requires both parties involved to be vigilant about keeping the symmetric key a secret. Let's say Alice and Bob are communicating via symmetric encryption. As long as no one else has the secret symmetric key, Alice has some certainty she really is communicating with Bob and vice-versa. But what if a third party, Lucy, were to obtain the symmetric key? Not only can Lucy pose as Bob or Alice (violating authentication and non-repudiation), she can also decrypt any messages sent by either party (violating confidentiality), as well as tamper with any message she chooses before sending it on its merry way or dropping it altogether (violating message integrity).

Asymmetric encryption, on the other hand, involves the use of a key pair: a public key and a private key:

```
Encryption:  plaintext --> Alice's public key --> ciphertext
Decryption:  ciphertext --> Alice's private key --> plaintext

Alternatively,

Encryption:  plaintext --> Alice's private key --> ciphertext
Decryption:  ciphertext --> Alice's public key --> plaintext
```

Figure 2. Encryption and decryption using public (asymmetric) keys.

These keys utilize cryptographic algorithms that, mathematically, are functional inverses of each other. Although the algorithms used to derive the keys are mathematically related, one of its inherent properties is that, given one of the keys, it is virtually impossible to derive the other. As stated by Dana Mackenzie in her article "The Code War":

> "The security of many public-key systems ... is explicitly based on the long-standing challenge in the branch of mathematics known as number theory ... Given a (very, very large) number obtained by multiplying two other numbers, find the two (very large) numbers that were multiplied to produce it. These numbers can be thought of as keys that lock and unlock the encryption code. The task of finding these keys is so difficult that snoops must either be bizarrely lucky ... or they must solve a problem that has stumped the smartest people in the world for more than 2000 years."[2]

As its name implies, the public key can made freely available. It can be distributed either via email, or by storing it on a publicly accessible directory. The private key, on the other hand, is kept secret. For example, if Alice and Bob were to communicate using asymmetric encryption, each of them would have two keys: a public key, which they would send to each other or store in a public place, and a private key, which each of them keeps secret and guards closely. The idea behind asymmetric encryption is that any data that is encrypted with the public key can only be decrypted with the private key, and any data encrypted with the private key can only be decrypted with the public key. Asymmetric encryption is slower than its symmetric counterpart, utilizing more resource overhead, hence it has a lower network throughput.

SSL uses asymmetric encryption only for authentication. According to Netscape's DevEdge article How SSL Works: "Netscape has licensed RSA public key cryptography from RSA Data Security Inc. for use in its products, specifically for authentication."[3] After both parties have been successfully authenticated, client and server agree upon a symmetric (shared) key, which is used for the remainder of the session (bulk data encryption).

Public Key Cryptography: A Closer Look

Many online discussions and tutorials depicting the how public key cryptography works use the names 'Alice' and 'Bob' to illustrate the two parties that wish to authenticate each other via PKI. I will maintain that convention in this paper. Before we delve into the details, however, let us take a moment to clarify the terminology we will be using in the discussion below.

---

[2] Dana Mackenzie, The Code War, http://www.beyonddiscovery.org/content/view.article.asp?a=3420
[3] How SSL Works, http://developer.netscape.com/tech/security/basics/index.html

Notation	Meaning
MSG	Alice's message to Bob
pub-key	Public key
priv-key	Private key
$A_{pub\text{-}key}$, $A_{priv\text{-}key}$	Alice's public key and private key, respectively
$B_{pub\text{-}key}$, $B_{priv\text{-}key}$	Bob's public key and private key, respectively
(MSG)	Parentheses around 'MSG' indicates encryption of MSG
{MSG}	Curly brackets around 'MSG' indicates decryption of MSG
$(MSG)A_{pub\text{-}key}$	MSG has been encrypted using Alice's public key
$\{MSG\}A_{pub\text{-}key}$	MSG has been decrypted using Alice's public key
MSG_H	Hashed Message
sym-key	Symmetric (shared) key

In describing how public key cryptography works, we are making the assumption that Alice and Bob already have each other's public keys. We are also not concerning ourselves, at least momentarily, with certificate validation. The main purpose of this section is to depict how two parties would use public key cryptography to: (1) authenticate each other; and (2) transmit a shared key.

Step 1: Alice has a message to send

Alice wants to send a confidential message, MSG, to Bob.

Step 2: Alice creates a *digital signature*

According to the FAQ discussions in the RSA Security website, "Sometimes one may want to verify the origin of a document, the identity of the sender, the time and date a document was sent and/or signed, the identity of a computer or user, and so on. A digital signature is a cryptographic means through which many of these may be verified."[4] Alice creates a digital signature in order to prove that she signed the message and also to ensure integrity of the message as it goes across the wire.

Essentially, a digital signature is a message digest that has been signed with a user's private key. To create a digital signature, Alice puts MSG through a hash function (normally SHA-1 or MD5), resulting in MSG_H. A hash is a one-way function which, when applied towards plaintext, produces an indecipherable message digest. A hash function can accept a message of any length, and will produce a digest that is much shorter and fixed in length. Since the hash is a one-way function, the resulting message digest cannot be reversed to produce the original message.

[4] RSA Laboratories, Cryptography FAQ, "What is a digital signature and what is authentication?" http://www.rsasecurity.com/rsalabs/faq/2-2-2.html

Alice encrypts MSG_H with her private key. The result is a hashed message which has been digital signed by Alice (using her *private key*). That's all a digital signature is. Depicted another way, this is what creating a digital signature looks like (where hash_function would correspond to a hashing algorithm like SHA-1 or MD5):

$$MSG \rightarrow \text{<hash\_function>} \rightarrow MSG_H \rightarrow (MSG_H)A_{priv\text{-}key}$$

Graphically depicted, this is what Alice's digital signature looks like:

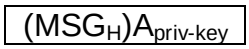


Figure 3: Alice's digital signature

As mentioned above, Alice creates a digital signature to accomplish two things: (1) to prove that she signed the message and (2) to prove the integrity of the message. The way this is done will become clearer as we go through the remainder of the steps, but as a heads up, since she signed the message with her private key, that means it can only be decrypted with her public key. That means, when Bob (later on down the line) decrypts the digital signature using Alice's public key, he will know that the message must have come from Alice because he was able to decrypt it successfully using her public key. Next, the reason hash functions can prove message integrity is because Bob also knows what hash function to use for messages sent by Alice. Later on, when he receives the plaintext version of MSG, he can run a hash through it and see if the message digest matches what Alice sent. If it matches (i.e., checksum comes out identical) then he knows the message has not been tampered with. Therefore, because of that, we can now see how digital signatures accomplish two things: authentication and verification of message integrity. It should be noted that since Bob can authenticate that Alice sent the message, this also accomplishes non-repudiation (if Bob can prove that Alice sent the message, she cannot say at some later time that the message didn't come from her).

At this point Alice actually attaches her message, MSG, to her digital signature. She then encrypts the entire message with Bob's public key and sends it to Bob.

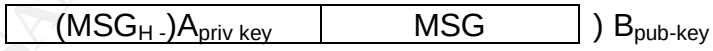
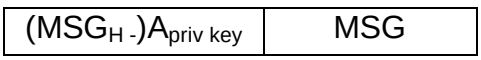


Figure 4: Alice's digital signature with message attached, encrypted with Bob's public key

Step 3: Bob receives Alice's message

When Bob receives Alice's message, the first thing he does is to decrypt the message with his private key. This will result in Alice's digital signature with MSG attached:



digital signature plaintext message

Next Bob ensures the message truly came from Alice by extracting her digital signature, $(MSG_H)A_{priv\text{-}key}$. Bob already has Alice's public key, so he can use that to decrypt the hashed message. Bob extracts Alice's digital signature from the packet, uses her public key to decrypt it, and ends up with the message digest:

$$\{(MSG_H)A_{priv\text{-}key}\}\ A_{pub\text{-}key} \text{ --> } MSG_H$$

Step 4: Bob creates his own hash of MSG

In the message that Bob received from Alice was a plaintext version of MSG. At this point both Alice and Bob know which hashing algorithm (aka "message authentication code", or "MAC") to use because they have already agreed upon it (i.e., as part of the SSL Handshake, which is discussed below). Bob now needs to ensure the integrity of the message he has received. He does this by running MSG through a hash function (the same one Alice ran MSG through). If the message has not been tampered with, the resulting hash will be identical, bit for bit, to the hashed message that Alice sent him (which was sent with the digital signature). If the message has been tampered with, Bob will detect a discrepancy between the two hashed messages, and the conversation will be terminated.

Graphically depicted, here is what Bob does:

$$MSG \rightarrow \text{<hash\_function>} \rightarrow MSG_H$$

If Bob's MSG_H = Alice's MSG_H, the integrity of MSG is intact, and the conversation between Bob and Alice can continue securely. That concludes the use of public key cryptography for authentication.

Using Public Key Cryptography to transmit the Shared Key

It was mentioned above that public key cryptography can also be used to transmit shared (symmetric) keys. This can be done in conjunction with the use of digital envelopes. According to RSA Laboratories, Cryptographic FAQ: "When using secret-key cryptosystems, users must first agree on a session key, that is, a secret key to be used for the duration of one message or communication session. In completing this task there is a risk the key will be intercepted during transmission. This is part of the key management problem .... Public-key cryptography offers an attractive solution to this problem within a framework called a digital envelope."[5]

Essentially, a digital envelope is a combination of two things: (1) a message that has been encrypted with a shared symmetric key; and, (2) a symmetric key that

[5] RSA Laboratories, Cryptography FAQ, "What is a digital envelope?", http://www.rsasecurity.com/rsalabs/faq/2-2-4.html

has been encrypted with another party's public key. The procedure for transmitting a shared key using public key cryptography and digital envelopes is as follows:

Step 1: Alice creates a random session key (aka, shared symmetric key, sym-key). How this is done will be further explained below, in the discussion of the SSL Handshake Protocol.

Step 2: Alice encrypts the message she wants to send to Bob with the symmetric key:

$$(MSG)_{sym\text{-}key}$$

Step 3: Alice encrypts the symmetric key (sym-key) with Bob's public key ($B_{pub\text{-}key}$):

$$sym\text{-}key \rightarrow (sym\text{-}key)\ B_{pub\text{-}key}$$

Step 4: Alice puts the output from Steps 2 and 3 together, and voila! She now has a digital envelope. Graphically depicted, this is what the digital envelope looks like (note that the left side is the encrypted message, the right side is the encrypted symmetric key):

Encrypted Message	Encrypted symmetric key
$(MSG)_{sym\text{-}key}$	$(sym\text{-}key)\ B_{pub\text{-}key}$

Figure 5: A digital envelope

Step 5: Bob reads the message by using his private key to decrypt the symmetric key.

$$\{(sym\text{-}key)\ B_{pub\text{-}key}\}\ B_{priv\text{-}key} \text{ --> } sym\text{-}key$$

Step 6: Bob reads MSG by using sym-key to decrypt it.

$$\{(MSG)_{sym\text{-}key}\}_{sym\text{-}key} \text{ --> } MSG$$

And that is how Bob and Alice can use public key cryptography to exchange symmetric keys. From the discussion above, we can see how public key cryptography is well suited to securely authenticating both parties in an SSL transaction.

SSL connections vs. SSL sessions

At this point we now turn our attention away from public/symmetric encryption and focus more on the actual SSL protocol.

An SSL connection is a transient, peer-to-peer relationship between two hosts within a network. Every connection is associated with a session; in fact, it is possible for more than one connection to be associated with one session. Connections associated with a session acquire that session's cryptographic security parameters.

An SSL session is an association between a client and a server created by the SSL Handshake Protocol. It is the SSL Handshake Protocol that defines the cryptographic security parameters that are valid for a given session. Sessions are created to avoid the resource overhead of having to renegotiate new cryptographic security parameters for each new connection.

SSL on the TCP/IP Protocol stack

In the TCP protocol stack, you will find SSL in between the Application and TCP/IP (Transport) layers, leveraging the capabilities of the TCP protocol to ensure reliable end-to-end connectivity. SSL is actually comprised of several sub-protocols, as depicted below: [6]

<table>
<tr><td>SSL Handshake Protocol</td><td>SSL Change Cipher Spec Protocol</td><td>SSL Alert Protocol</td><td>HTTP</td></tr>
<tr><td colspan="4">SSL Record Protocol</td></tr>
<tr><td colspan="4">TCP</td></tr>
<tr><td colspan="4">IP</td></tr>
</table>

Figure 6. The SSL Protocol Stack

SSL Record Protocol

The higher level SSL Protocols (SSL Handshake Protocol, SSL Change Cipher Spec Protocol, SSL Alert Protocol) use the SSL Record Protocol to transmit messages "in a secure manner with message integrity ensured". [7] The way it does this is as follows:

- The SSL Record Protocol takes an application message to be transmitted (from the higher level SSL protocols)
- The data is fragmented into sizes 16KB or smaller

[6] Information on how the SSL Protocols function was gleaned from four separate sources: http://134.193.15.25/vu/course/cs596A/lessons/topic20/SSL/sld011.htm, http://www.windowsecurity.com/articles/Secure_Socket_Layer.html, http://wp.netscape.com/eng/ssl3/3-SPEC.HTM#7-3 and http://mit.iddl.vt.edu/courses/cs5244/coursecontent/mod6/lesson3/sec23.html

[7] http://www.windowsecurity.com/articles/Secure_Socket_Layer.html

- At this point it is possible to compress the data. However, compression is not an option in SSL version 3.0; hence, this operation does not take place.
- A Message Authenticate Code (MAC) value of the application message is created and added to the data fragment.
- The application message and the MAC are encrypted using symmetric encryption.
- An SSL Record header is appended to the fragment.
- The unit is passed down the TCP protocol stack as a TCP segment.

Change Cipher Spec Protocol

This protocol consists of a single byte (value=1). Its purpose is to verify that future transactions for a given session will use the agreed upon CipherSpec and keys.

Alert Protocol

This protocol consists of two bytes. The first byte can take one of two types of values: warning message, or fatal message. If a fatal message is delivered, the SSL session is terminated. The second byte contains error codes.

Handshake Protocol

The Handshake Protocol initiates the session between client and server. This is where client and server authenticate each other, negotiate an encryption algorithm, a MAC algorithm and cryptographic keys to be used. The Handshake Protocol consists of four distinct phases:

Handshake Protocol Phase 1. The association between client and server is initiated, security settings are established. The client initiates Phase 1 of the Handshake Protocol by sending a "client_hello" message. The following parameters can be found in the client_hello message:

Parameter	*Meaning*
SSL version number	Highest version of SSL which the client supports
Cipher Suite	Key exchange methods and CipherSpec supported by the client. Examples are: RSA, Diffie-Hellman, Fortezza; RC4, RC2, DES, 3DES,
Random data	Random structure (combination of 32-bit timestamp and 28 bytes from a random number generator)
Session ID	Session identifier. A Session ID of 0 signifies a new session.
Compression	Compression method supported by client

Figure 7. Information sent by the client inside the "client_hello" message.

The server responds in kind with a server_hello message:

Parameter	*Meaning*
SSL version number	Lowest version of SSL supported by the client and the highest version supported by the server
Cipher Suite	Cipher Suite chosen by the server from the list of options presented by the client
Random data	Random structure (combination of 32-bit timestamp and 28 bytes from a random number generator)
Session ID	If client session ID = 0, server's session ID will be the value of a new session; else, server's session ID will be same as client's session ID
Compression	Compression method chosen by the server from the list of options presented by the client.

Figure 8. Information sent by the server inside the "server_hello" message.

Handshake Protocol Phase 2. In this phase the server sends the client its certificate (in which the client will find the server's public key). If the server does not have a certificate, it will send the client a server_key_exchange message, in which the client and server can create keys to use for future communication. If the server requires client authentication (which is optional with SSL), it is at this point that the server would request certificate information from the client. At the end of Phase 2, the server sends the client a server_done message, then it waits for the client's response.

Handshake Protocol Phase 3. The client authenticates the server by verifying information received (from the server). If the server requested a client certificate, it is at this point that the client would send its certificate back to the server. If the client successfully authenticates the server, it generates a pre-master secret key. It uses a temporary RSA key from the server_key_exchange message *OR* the server's public key from the certificate to encrypt the pre-master secret.

Handshake Protocol Phase 4. Client sends the server a change_cipher_spec message, which establishes the agreed-upon cryptographic settings as the current configuration for the session. In response, the server will also send a change_cipher_spec message to the client. This concludes the handshake, and the session is closed. However, the session state is left open so that communication could resume at any time using the agreed-upon parameters.

Generation of the Symmetric Key[8]

Both server and client have the same premaster secret (since the client sent the server the premaster secret in Handshake Protocol Phase 3). They both process

[8] The information on how symmetric keys are generated were obtained from: Introduction to SSL, http://docs.sun.com/source/816-6156-10/contents.htm

the premaster secret through a hash algorithm to generate a master secret. The client and server then respectively process the master secret to generate session keys. Note that both server and client and perform these steps separately. They each end up with their own session keys, which are synonymous to symmetric keys. The symmetric keys are used to encrypt/decrypt data during the SSL session.

The client then sends a message to the server with the session key to be used for the remainder of the SSL session (for data generated by the client). The client also sends a separate encrypted message to the server indicating that the client portion fo the SSL handshake is complete. The server does the same; it sends a message to the client with the session key to be used for the remainder of the SSL session (for data generated by the server), as well as sends a separate message to the client indicating the server piece of the SSL handshake is also complete.

How SSL Uses Certificates

Certificates are a way of binding a public key to a name. According to DevEdge's Online Documentation: "A certificate is an electronic document used to identify an individual, a server, a company, or some other entity and to associate that identity with a public key. Like a driver's license, a passport, or other commonly used personal IDs, a certificate provides generally recognized proof of a person's identity."[9] For instance, if Alice wants to communicate with her online bank (let's call it "E-Bank") via SSL, how does she know it's really E-Bank she's talking to? Certificates provide a means for Alice to consult with a trusted third party (aka, "trusted CA") to verify that E-Bank really is whom it says.

How a certificate is authenticated

When a certificate is being authenticated, the authenticating party needs to verify four items[10]: (1) the certificate has not yet expired; (2) they need to check whether the CA that signed the certificate is a CA that they trust; (3) to make sure the certificate has not been tampered with, the CA's digital signature is also verified; finally, (4) the verifying party needs to make sure that the server's domain name and its DN value on the certificate are the same:

(1) Verifying that the certificate has not expired

[9] http://docs.sun.com/source/816-6154-10/
[10] The information on how certificates are authenticated was derived from http://docs.sun.com/source/816-6156-10/contents.htm

The client checks the validity period in the certificate and ensures that it has not expired.

(2) Verifying that the signing CA is a trusted one

If Alice wants to communicate with E-Bank, she initiates the conversation with the E-Bank server (i.e., via her browser). What she will get from E-Bank is a certificate. The certificate will be signed by a certificate authority ("CA"). In her browser, Alice maintains a list of CAs that she trusts. If you're using Internet Explorer, you can find the list of trusted CAs by selecting Tools --> Internet Options --> Certificates --> Certificates.

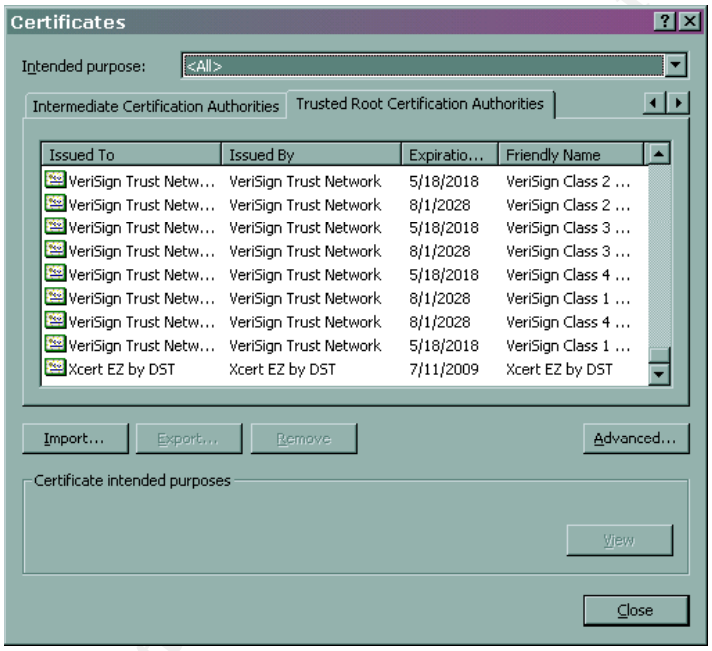


Figure 9. List of trusted CAs stored on an Internet Explorer browser

If the CA that signed E-Bank's certificate is on the list of Alice's trusted CAs, she can move on to Step 3.

(3) Verifying that the certificate has not been tampered with

The certificate from the server contains a digital signature signed by the certificate issuer. Recall that a digital signature is a message digest that is signed by a user's private key. If the CA is on the list of Alice's trusted CAs, then she has the CA's public key stored in her repository (on her browser). She uses that public key to decrypt the CA's digital signature. She also verifies that the server certificate hasn't been altered since the digital signature was signed. If that is successful, then she knows that the cert is valid. She can then move on to the next step.

(4) Verifying that the server's domain name and its DN value on the certificate are the same

This step is not technically part of the SSL handshake, but as part of a security precaution, the client needs to determine that the server's domain name and the DN value on the certificate are identical.

(5) Client authenticates server certificate

Assuming all the previous steps completed successfully, the client can now authenticate the server certificate.

By authenticating the server's certificate, the client now has some assurance that she is communicating with the correct entity, and the SSL connection can proceed as usual.

Conclusion

That sums up our discussion of SSL. We saw where SSL resides on the TCP/IP protocol stack, we discussed the types of encryption SSL uses, the process behind public key cryptography and the SSL Handshake. We saw the role that certificates play in the SSL handshake, as well as how the client can authenticate a server certificate. SSL is a wonderful tool for us to have today; without it, all E-Commerce transactions would be impossible. This all happens under the covers, without any need for manual intervention from the end user; however, it would benefit everyone to have some understanding of how SSL works and how it protects their information. As this paper illustrates, there are many components to SSL; it is my hope that this discussion was able to provide a clear overview of SSL's various parts and how they interact with each other.

References

Onyszko, Tomasz . Secure Socket Layer. 19 July 2002 URL: http://www.windowsecurity.com/articles/Secure_Socket_Layer.html (15 Oct. 2003)

Harn, Lein. Secure Web Applications – SSL. 22 December 2000 URL: http://134.193.15.25/vu/course/cs596A/lessons/topic20/SSL/sld001.htm (15 Oct. 2003)

Oliva, Mark. Lesson 3: Secure Sockets Layer. http://mit.iddl.vt.edu/courses/cs5244/coursecontent/mod6/lesson3/sec23.html (19 Nov. 2003)

Netscape Communications Corporation. Introduction to SSL. 09 October 1998 URL: http://docs.sun.com/source/816-6156-10/contents.htm (19 Oct. 2003)

Mackenzie, Dana. The Code War. February 2003. URL: http://www.beyonddiscovery.org/content/view.article.asp?a=3420 (01 Oct. 2003)

Internet Draft SSL 3.0 Specification. Work-in-progress. URL: http://wp.netscape.com/eng/ssl3/3-SPEC.HTM#7-3 (19 Nov. 2003)

Netscape Communications Corporation. How SSL Works. 1999 http://developer.netscape.com/tech/security/basics/index.html (01 Oct. 2003)

RSA Laboratories. RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1. 2000. URL: http://www.rsasecurity.com/rsalabs/faq/index.html (01 Oct. 2003)

Netscape Communications Corporation. Secure Sockets Layer. 2003 http://wp.netscape.com/security/techbriefs/ssl.html (01 Oct. 2003)

Verisign, Inc. Introduction to Public Key Cryptography. http://www.verisign.com/repository/crptintr.html (01 Oct. 2003)

Netscape Communications Corporation. Introduction to Public-key Cryptography. 1998. URL: http://docs.sun.com/source/816-6154-10/ (19 Nov. 2003)

Last Updated: July 24th, 2017

Upcoming Training

SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor