The SSL Handshake

A Perspective on Practical Security 2004
by Brad C. Johnson, Jonathan G. Gossels, & Donald T. Davis

© Copyright 1997-2004 SystemExperts Corporation. All rights reserved.
The SSL Handshake

Abstract

SSL is arguably the Internet’s most widely-used encryption system. Increasingly, SSL is used not only for secure Web connections, but for all manner of network applications that need end-to-end encryption. Application developers often find SSL complex and intimidating, but programmers should not rely only on SSL’s simplest modes of operation, because both security and performance will suffer.

While SSL use is now somewhere between ubiquitous and conventional, most application developers and network professionals understand little about how it really works. Performance, latency, and scalability of solutions are all impacted by the way SSL operates. This paper addresses only one part, but arguably the very essence of SSL, the SSL Handshake. That is, what is this handshake used for, when is it used, and what are the key implications for the applications that use it?

Inside

- Why is understanding the SSL handshake important to the design of your web application?
- When does the SSL handshake happen?
- What happens during the SSL handshake?
- What are the main functions of SSL?
- What types of authentication, cipher, and hash methods are available?
- How to find out more about SSL.

Context

Where does SSL fit?

The SSL handshake impacts Web application usage and design; one must keep in mind where it resides and how it is normally used. Foremost, SSL (as defined in its name) is a socket-layer interface. In the network stack model, this means that SSL is in below the application/presentation layer and above the transport layer. Contrary to what many people think, SSL is not wedded to HTTP. Any application level program — not just HTTP — can run within an SSL connection! Nevertheless, HTTP is the protocol that runs inside SSL most frequently and so it is only natural to think of them together.

The SSL protocol provides communications privacy over the Internet. SSL doesn’t support file-encryption, access-control, or virus protections, so SSL cannot help manage sensitive data before and after secure transport. The protocol is specialized to allow clients and servers to communicate without suffering from impersonation, eavesdropping, tampering, or message forgery. As with any encryption system, there is an aspect of key management inherent in SSL’s use.

The IETF has embraced SSL as the main part of the Transaction Layer Security (TLS) standard. Though TLS is based upon SSL, TLS extends SSL in several ways, and the two protocols don’t interoperate directly. A notable difference between SSL and TLS is that TLS supports other kinds of key-negotiation besides SSL’s public-key handshake. Overall, TLS restores some of the crypto-choice flexibility (and complexity) that has become obsolete since SSL’s early days.

Why is it used?

In every SSL session, the server (responding) side of the connection MUST authenticate itself to the client (requesting) side. The threat is that a client may ask for a particular service but the reply to that request may come from an impostor or may be observed by an eavesdropper. To counter the possibility of an impostor server reply, the server proves its identity to the client. In SSL V3.0, the server MAY
The SSL Handshake

also require the client to authenticate himself to the server, a result commonly known as “mutual authentication.” SSL performs authentication with public-key cryptography, and protects confidentiality and message-integrity with symmetric-key cryptography.

After the initial authentication, SSL negotiates a set of cipher choices for the symmetric-key protections. Because the client and server can prefer different ciphers, this negotiation guarantees that the client and server both will use the strongest crypto algorithms that they’re both able to use. Unfortunately, SSL offers many more optional ciphers than anyone needs; making SSL’s configuration seem much more complicated than it really is.

Most SSL servers, browsers, and other applications achieve very good security with the following crypto algorithms:

- Public-key cipher: RSA
- Hash function: SHA-1 or HMAC-SHA-1 (avoid MD5)
- Symmetric-key cipher: AES or 3DES
  - Always use CBC mode for block ciphers
  - Avoid single-key DES
  - Avoid 40-bit ciphers

SSL’s configuration will also ask you to choose key-lengths for these ciphers. It’s important to choose the three algorithms’ key-lengths so that all three have the same effective strength. For more information, see “Equivalent Algorithm Strengths” in NIST’s Key Management Guideline.

What’s to worry about?

One of HTTP’s important design features is its statelessness. A server for a stateless protocol doesn’t try to remember anything about its clients, so the server can be replicated for load management, and can fail and restart, all without coordination with the client side. In choosing a stateless protocol, HTTP’s designers avoided many of the complexities of a network-based transaction environment. But, SSL is not stateless, and this complicates SSL’s performance trade-offs.

The most significant side effect of HTTP’s statelessness arises because most page-loads can require substantial connection overhead. For example, to read a page with four pictures, HTTP/1.0 requires five separate TCP connections, one for each HTTP GET request. With SSL, this seems to imply a costly SSL handshake for each and every one of these GET connections — a real kick in the head for performance and scalability. Luckily, for most applications it’s not necessary to redo the handshake repeatedly.

Let’s take a closer look.

The SSL Handshake

To paraphrase from the SSL specification, the goal of the SSL handshake between the client and the server is:

- to negotiate an acceptable SSL version. i.e., V2 or V3,
- to select the appropriate cipher and hash methods,

---

5. Early versions of SSL supported many cipher alternatives, because the best ciphers and the fastest implementations were usually expensive in one way or another. Some were patented, others were export-controlled, and still others were new and risky. Now that strong ciphers are available without royalties or export-controls, SSL users only need to worry about trading off performance against security by selecting key-sizes carefully.


8. In a stateless protocol, only the server is stateless. Each client remembers whatever the server needs to know about the client’s session, and reminds the server as necessary.

9. By contrast, HTTP/1.1 can gang all five GET operations in a single TCP/IP connection.

10. This is the source of some confusion about Web site “hits.” It’s often reported that a particular site receives, say, 100,000 hits per day. Well, if the home page for that site contains 20 different video, audio, and graphic objects (common for many high-profile sites), “100,000 hits” means that 5,000 visitors actually referenced the home page. If the site is a news page or a weblog, its users might revisit the site 2 or 3 times a day. So, only 1000 unique users of the site loaded the home page — not exactly what you’d think from hearing that the site had 100,000 hits!
The SSL Handshake

› to authenticate the server (and the client if necessary), and
› to securely negotiate shared secrets.

While there is overhead associated with all of these functions, by far the most costly is the authentication phase, because this step uses one or more public-key encryptions and signatures.11

The following is the logical flow of the SSL handshake, c.f.,
ftp://ftp.isi.edu/in-notes/rfc2246.txt

<table>
<thead>
<tr>
<th>Step</th>
<th>Client/Server</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Establish connection</td>
<td>If a resumed session, GOTO step III; Else, establish protocol version, session ID, cipher suite, compression method, and client/server random values</td>
</tr>
<tr>
<td>II</td>
<td>Exchange certificates or keys</td>
<td>Authentication and Signing, i.e., X.509 certificate and/or key exchange and validation; Uses expensive public key encryption &amp; signature</td>
</tr>
<tr>
<td>III</td>
<td>(Re)Establish session and connection state</td>
<td>Save session and connection state information for resuming future connections (more) quickly; Uses inexpensive hashing operations.</td>
</tr>
<tr>
<td>IV</td>
<td>Handshake Completion</td>
<td>Confirm client/server key agreement before application data is transferred</td>
</tr>
</tbody>
</table>

By default, the handshake is performed every time the client makes a TCP connection to the server. A single page download can entail numerous connections, because most HTML documents contain a mix of multiple objects, including HTML text, JPEG/JPG/GIF/BMP pictures, QT/MOV movies, MPEG/MPG video, RAM audio, PDF documents, etc. To avoid much of this overhead, most deployed servers support the notion of a resumed session, that is, even though HTTP is a stateless protocol, SSL is not. SSL maintains two different types of state: session state and, within that, per-connection state. Each client and server party may have multiple sessions and each session may have multiple connections. Each connection has unique keys, whether the session is a new or resumed one, but the expensive public key operations happen only once per session.

The server side must retain current information associated with these different states. If an existing session cannot be resumed, e.g., the server cannot find the sessionID offered by the client in its cache, then a full handshake must be initiated. No application data can be transmitted until this handshake is complete, hence server failure cannot be made invisible and load balancing amongst SSL servers can be no finer grained than one session per server.

SSL Overhead

Unfortunately, it is non-trivial to predict SSL performance impact on an application, because the impact depends on several critical variables such as:

› Which version of SSL does your server use?
› How long are your public key pairs?
› Which cipher and hash are being used?
› How many times must the handshake be performed?
› How thoroughly do certificates get checked?12
› How efficient is the client’s SSL implementation?13

There are ways to account for these variables. Most organizations define exactly what types of key management, ciphers, and hashes they prefer a priori. Therefore, a specific (and knowable) amount of overhead can be associated with a normal transaction; e.g., for your Web application you may elect to use “AES encryption with a 128-bit key and an

11. Each handshake costs the SSL server 5-6 msec of CPU-intensive bignum arithmetic: Basic SSL always causes the server to decrypt a public-key ciphertext (5 msec), and if the client presents a certificate, the server must also validate two signatures (each takes 0.5 msec).
12. While certificate revocation and Root-key validation are not yet implemented outside the laboratory, their adoption will be driven by B2B concerns about financial risk.
13. Though 95% of browsers use Microsoft’s MSIE SSL implementations, Microsoft’s different OS versions vary in their SSL efficiency, and non-browser applications often use other SSL libraries, anyway.
The SSL Handshake

SHA-1 MAC.” Therefore, this combination, and the overhead it introduces, might be a constant within the context of your application.

Using this logic, the key variables include:

› Are resources outside the controlled process flow involved (such as calls to an external Certificate Authority)?
› How many times will the handshake be used?

If you are using standard Web client/server kits such as Apache, Microsoft, and SunONE, page design and transaction flow determine how many times the handshake will be used; the leading Web servers are pretty much the same in their SSL initiation overhead.14 For other SSL libraries, such as OpenSSL, Java, Jetty, and application servers like a DBMS, the timing behavior will vary somewhat, but in every implementation, the handshake is so expensive that applications should almost always use resumable sessions, instead of running the handshake repeatedly.

Even when a server uses resumable sessions, the SSL handshake will be the bottleneck that limits the server’s secure-session throughput. To improve SSL performance further, you should consider SSL accelerator products. Both hardware and software accelerators are available. Some software accelerators offer a tenfold speedup, and some crypto hardware vendors claim a 100-fold improvement. Some SSL accelerators work well with load balancers.

When Does the Handshake Happen?
The SSL handshake happens every time a client starts a session with a “new” server or when either party for whatever reason, including no good reason, wishes to establish a new SessionID and Cipher Specification.

The SSL handshake is normally NOT performed on every GET of an object so, for example, a page with four pictures and two videos would likely be one handshake and seven file opens.15 Since symmetric-key encryption is much faster16 than the handshake’s public-key operations, a multi-screen protected transaction should stay within a single SSL session, even if this means that some non-confidential pages are encrypted unnecessarily. Almost all SSL applications should use these non-default resumable SSL sessions.

Conclusion

SSL was designed to easily provide server authentication to clients and to provide efficient encryption negotiation when used with a protocol like HTTP that performs many small connections. Some of SSL’s flexibility is unnecessary; because SSL supports many legacy ciphers that are now largely irrelevant. The valuable parts of SSL’s flexibility are its client authentication options, and its multiple-connection resumable sessions. Besides being flexible, SSL is ubiquitous and well-studied, so it is an easy, off-the-shelf answer for the network security needs of most applications.

---
14. SSL implementations can vary greatly in their certificate-handling, because the X.509 certificate format is imperfectly standardized.
15. 7 downloads = 1 (page.html) + 4 (pictures) + 2 (videos)
16. Extra SSL handshakes slow a web-server by 4x, while AES encryption actually runs faster than the I/O bus.