Analysis of the SSL Protocol

Jeremy Bradley   Neil Davies

June 1995   **CSTR-95-021**

University of Bristol

Department of Computer Science

Also issued as ACRC-95:CS-021

Analysis of the SSL Protocol

Jeremy Bradley [1]
Department of Computer Science
University of Bristol

Neil Davies [2]
Department of Computer Science
University of Bristol

June 28, 1995

[1]E-mail: bradley@cs.bris.ac.uk
[2]E-mail: neil.davies@bristol.ac.uk

Abstract

SSL, as a proposed secure communications protocol, is set to become an international standard. As such it is important to have a full discussion of the technical and practical issues behind the protocol before the onset of commercial and industry acceptance. The protocol goes in front of an IETF BOF in July of this year and it is hoped that this document will be one of many to provide a small discussion point for that meeting.

This report analyses the communications structure of the SSL Handshake Protocol and suggests alterations based on such analysis. Other recommendations cover issues of general security, as well as some of the administrative points associated with certification.

Contents

1 Introduction

This is a technical report on the SSL Protocol as published by Netscape Communications Corporation [HIC95]. It represents high level analysis of the basic Handshake Protocol and does not touch on the lower level Record Protocol.

It is hoped that this document will be instrumental in the development of SSL as a thoroughly secure international standard and some of the recommendations supplied may be considered in future versions of the protocol.

Further it is intended to advance this project by producing a mathematical specification of a specific implementation with a view to creating a non-US *reference implementation* of the SSL protocol.

1.1 Message Dependancy Analysis

Message Dependancy Analysis serves three important purposes:

1. Highlights the non-determinism in a protocol, where messages maybe more loosely constrained and can therefore occur in any of a number of places in a message stream.
2. May point out omissions in a protocol. This often happens in special cases where a particular piece of information is requested but no corresponding error message is supplied as a possible response.
3. Can be used to locate message redundancy, where a particular information exchange takes place which is deemed to be unnecessary.

It is unlikely that a high level study, such as this, will point out any message redundancy and the process is being used primarily for the first two purposes.

1.2 Finite State Diagrams and Process Algebra

As a natural evolution of Message Dependancy Analysis, Finite State Diagrams [HAL93] can be produced for generic client/server operation. Generic Finite State Diagrams model every possible operation of any protocol compliant implementation. This is deduced from the Message Dependancy Analysis which has already allowed us to generate every possible exchange of messages. A generic Finite State Diagram can therefore be used as a basis for a specific implementation to communicate with any other protocol-compliant implementation, with the knowledge that it can never receive a message which it was not expecting.

Process Algebra is a formal description language for concurrent systems and is particularly useful for explicitly representing Finite State Diagrams. There also exist concurrency toolkits which can use Process Algebra to search for problems such as deadlock.

1.3 Recommendations

The *Recommendations* section represents a critique of the SSL Protocol based on the preceding analysis and other experience. It is realised that some of the comments on certification are outside the particular realm of the SSL Protocol but they were included as they were relevant to the security of the protocol.

2 Message Dependency

Dependency analysis is a useful tool in examining communication protocols as it can be used to show the amount of allowable non-determinism within a protocol. In practical terms, it can show which combinations of messages comply with the protocol and which do not.

Message B is **data dependent** on another message A if message B contains some information that is provided by message A. This means that message A must be received (or transmitted) before message B is transmitted (or received) for a valid exchange to occur. If this ordering is not maintained then a protocol violation error message may be generated.

Temporal dependency is a slightly weaker constraint and is usually specified explicitly in a protocol definition. As an example, a protocol might define an opening message A and a closing message Z in a particular exchange between which all other messages must occur. Messages B—Y must occur after A and before Z even though some of the messages may not have any direct data dependency on A and similarly Z may not have any direct dependency on any of B—Y.

The notation below is commonly used to represent dependency of any kind:

$$A \longrightarrow B \quad \equiv \quad B \text{ `is dependent on' } A$$

Note that this relation is transitive so the rule below follows:

$$\text{if} \quad A \longrightarrow B \quad \text{and} \quad B \longrightarrow C \quad \text{then} \quad A \longrightarrow C$$

This transitivity can be used to significantly reduce the complexity of such diagrams since if all the relations $A \longrightarrow B$, $B \longrightarrow C$ and $A \longrightarrow C$ are displayed on a diagram then $A \longrightarrow C$ can be removed since it is already implied by the other two.

3 Dependency Analysis of the SSL Protocol

The message dependency diagrams in figures 1—4 represent the data and temporal dependencies of the various possible messages defined in the SSL Handshake Protocol. The presentation of these diagrams helps in the construction of Finite State Diagrams and thus a Process Algebra description of both client and server – the goal of this stage of project design.

The diagrams have been simplified using the transitivity law above. Also an informal convention has been used – that of dashing dependency arrows. This indicates that some messages are conditional and may only occur under certain circumstances – where they do occur they are constrained by the dependency shown. This is particularly useful when modelling error states.

The SSL protocol also defines a 2 stage process where in general messages in the first stage cannot be interleaved with messages in the second – this feature has also been marked on the diagrams.

There are 4 main types of a SSL handshake depending on the circumstances and application configuration:

1. No server cached session-id and no client authentication
2. No server cached session-id and client authentication used
3. Server and client cached session-id found and no client authentication
4. Server and client cached session-id found and client authentication used

3.1 No server cached session-id, no client authentication

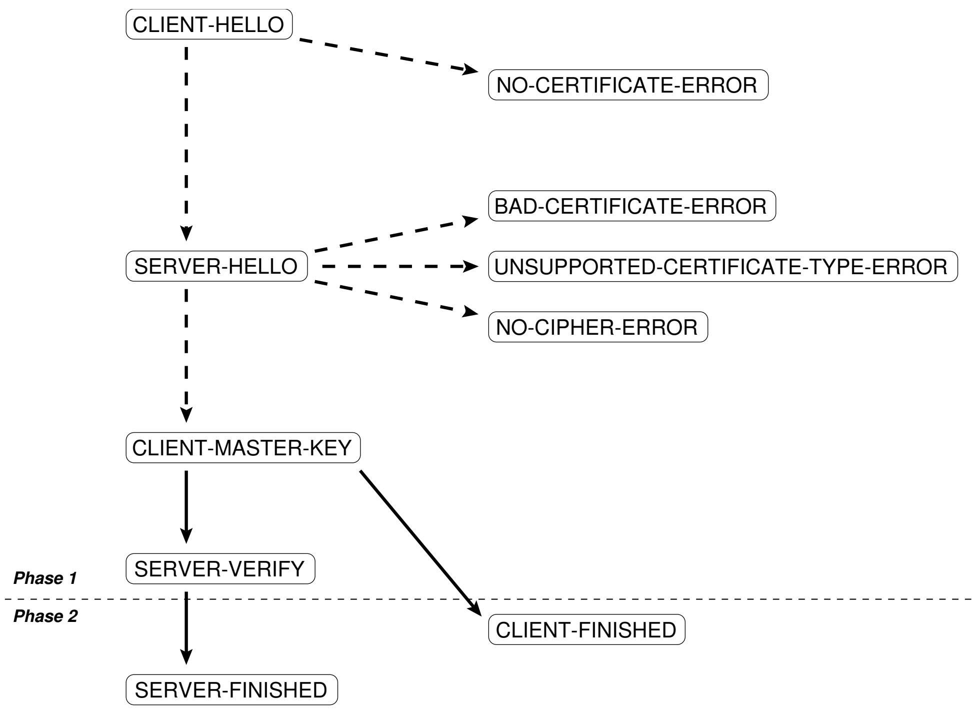

Figure 1: No cached session-id, no client authentication

Figure 1 is for an initial-session exchange and shows all the certification error messages that the client can produce in response to the server certificate in the SERVER-HELLO message. Since this is an initial message the CLIENT-MASTER-KEY has to be exchanged at this stage too.

As an example of how figure 1 can be used to deduce allowable message sequences, the following are all valid:

- [CLIENT-HELLO, SERVER-HELLO, CLIENT-MASTER-KEY, CLIENT-FINISHED, SERVER-VERIFY, SERVER-FINISHED]
- [CLIENT-HELLO, SERVER-HELLO, CLIENT-MASTER-KEY, SERVER-VERIFY, CLIENT-FINISHED, SERVER-FINISHED]
- [CLIENT-HELLO, SERVER-HELLO, CLIENT-MASTER-KEY, SERVER-VERIFY, SERVER-FINISHED, CLIENT-FINISHED]
- [CLIENT-HELLO, SERVER-HELLO, NO-CIPHER-ERROR]

In particular the CLIENT-HELLO message can occur at any time after the CLIENT-MASTER-KEY message and still conform with the protocol.

3.2 No server cached session-id, client authentication used

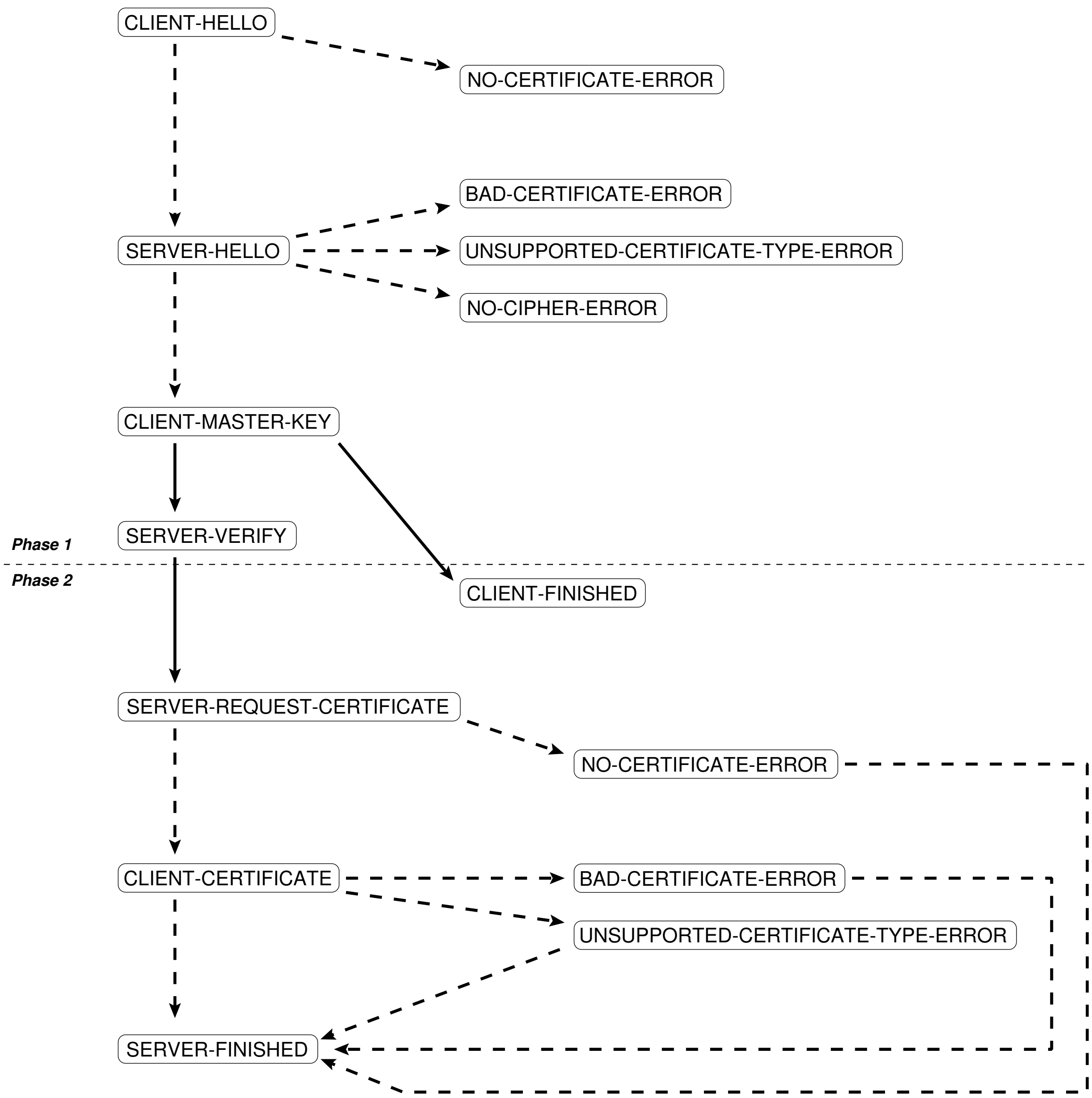

Figure 2: No cached session-id, client authentication used

Figure 2 is for an initial-session handshake and also includes the possible errors after a request for client certification, which may be issued by the server.

Some of the behaviour after errors occur is undefined and left up to individual implementations. In particular the certification errors produced after a REQUEST-CERTIFICATE message can either close the connection or leave it open.

A secure implementation would cause the connection to be closed after a BAD-CERTIFICATE-ERROR (sent by the server) and allow the behaviour of a server after the other errors to be defined by the application protocol. For instance a secure system would require the closing of the connection after any error.

3.3 Server/client cached session-id found, no client authentication

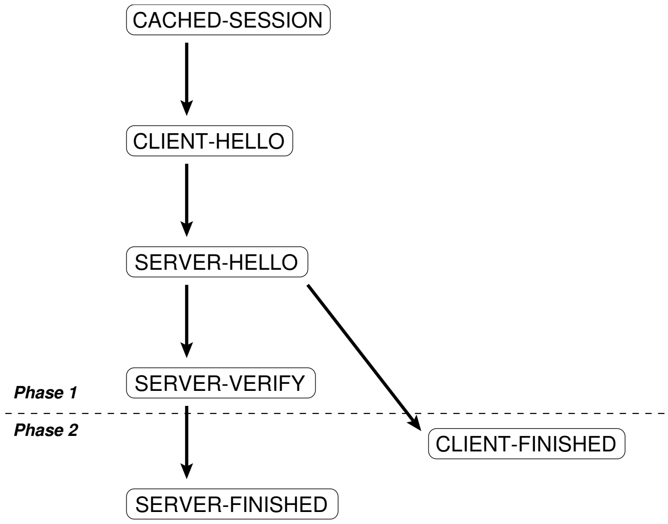

Figure 3: Cached session-id found, no client authentication

Figure 3 represents the first of the cached sessions. In this scenario the server certificate is not exchanged in the SERVER-HELLO message and the CLIENT-MASTER-KEY is remembered from the initial session.

3.4 Server/client cached session-id found, client authentication used

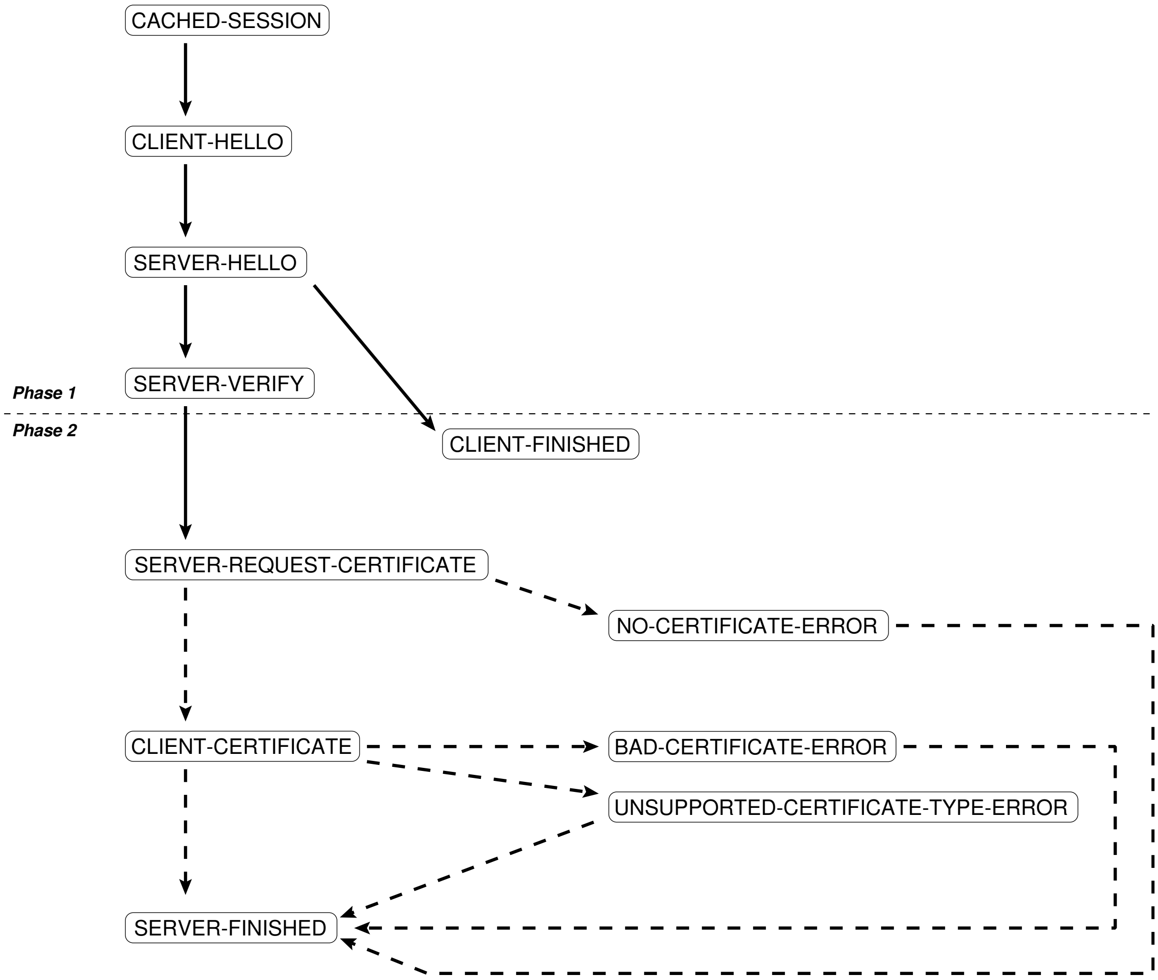

Figure 4: Cached session-id found, client authentication used

Figure 4 is another mid-session handshake – similar to figure 3. This diagram also demonstrates how client certification fits into a cached session handshake.

4 Client-Side Handshake Analysis

This section follows on from the message dependency analysis in that it derives finite state diagrams for each section of client negotiation. Following each FSD is a process algebra description of the diagram. These descriptions still represent a generic client that complies with the SSL protocol – that is, every compliant implementation of a client will lie within a subset of these descriptions.

Of particular note in this section is the explicit representation of two types of CLIENT-HELLO and SERVER-HELLO message namely, CLIENT-HELLO(SID), CLIENT-HELLO(NO-SID) and SERVER-HELLO(HIT), SERVER-HELLO(NO-HIT). These represent the possible states of both the client server cache and thus easily deal with situations where a session-id has been timed out and discarded.

4.1 Initial Client Negotiation Phase

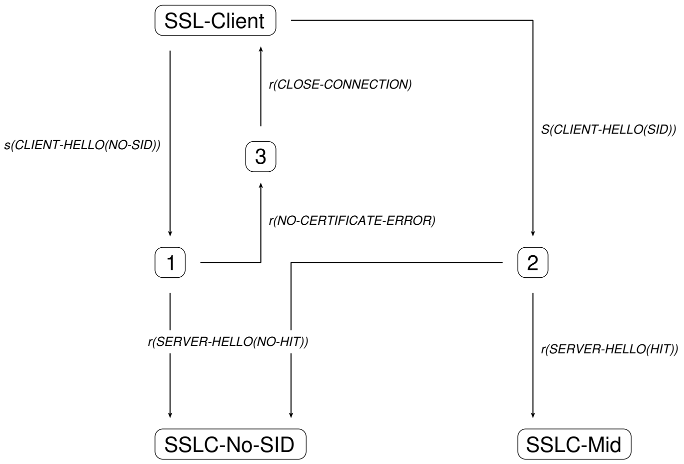

Figure 5: Opening client messages

$$
\begin{aligned}
\text{SSL-Client} &\stackrel{\text{def}}{=} \overline{\textsc{client-hello}(\textsc{no-sid})} \cdot \text{SSLC1} + \overline{\textsc{client-hello}(\textsc{sid})} \cdot \text{SSLC2} \\
\text{SSLC1} &\stackrel{\text{def}}{=} \textsc{server-hello}(\textsc{no-hit}) \cdot \text{SSLC-No-SID} + \textsc{no-certificate-error} \cdot \text{SSLC3} \\
\text{SSLC2} &\stackrel{\text{def}}{=} \textsc{server-hello}(\textsc{no-hit}) \cdot \text{SSLC-No-SID} + \textsc{server-hello}(\textsc{hit}) \cdot \text{SSLC-Mid} \\
\text{SSLC3} &\stackrel{\text{def}}{=} \textsc{close-connection} \cdot \text{SSL-Client}
\end{aligned}
$$

4.2 Master Key Negotiation

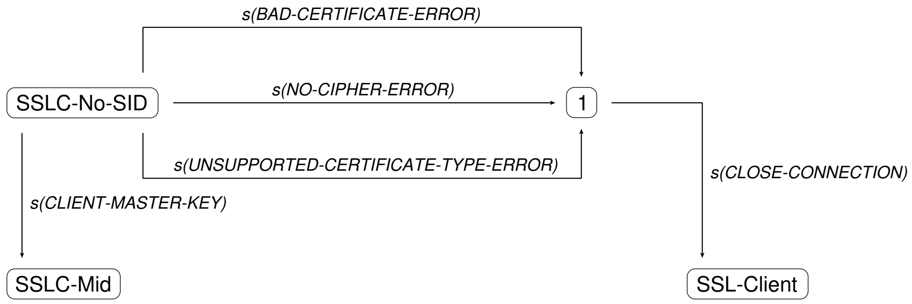

Figure 6: Client, if no session-id cached

$$
\begin{aligned}
\text{SSLC-No-SID} &\stackrel{\text{def}}{=} (\overline{\textsc{bad-certificate-error}} + \overline{\textsc{unsupported-certificate-type-error}} + \\
&\quad \overline{\textsc{no-cipher-error}}) \cdot \text{SSLCNS1} + \overline{\textsc{client-master-key}} \cdot \text{SSLC-Mid} \\
\text{SSLCNS1} &\stackrel{\text{def}}{=} \overline{\textsc{close-connection}} \cdot \text{SSL-Client}
\end{aligned}
$$

4.3 Middle Phase Client Negotiation

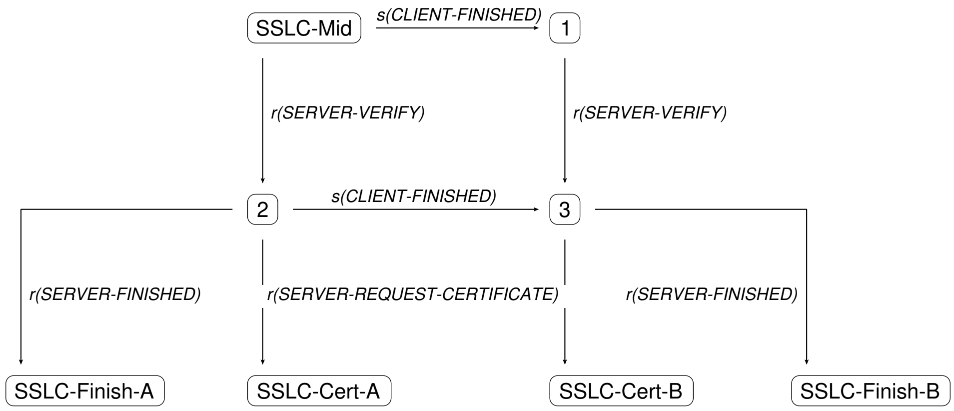

Figure 7: Client, next phase

$$
\begin{aligned}
\text{SSLC-Mid} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLCM1} + \textsc{server-verify} \cdot \text{SSLCM2} \\
\text{SSLCM1} &\stackrel{\text{def}}{=} \textsc{server-verify} \cdot \text{SSLCM3} \\
\text{SSLCM2} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLCM3} + \textsc{server-finished} \cdot \text{SSLC-Finish-A} + \\
&\quad \textsc{server-request-certificate} \cdot \text{SSLC-Cert-A} \\
\text{SSLCM3} &\stackrel{\text{def}}{=} \textsc{server-finished} \cdot \text{SSLC-Finish-B} + \\
&\quad \textsc{server-request-certificate} \cdot \text{SSLC-Cert-B}
\end{aligned}
$$

4.4 Client Certification

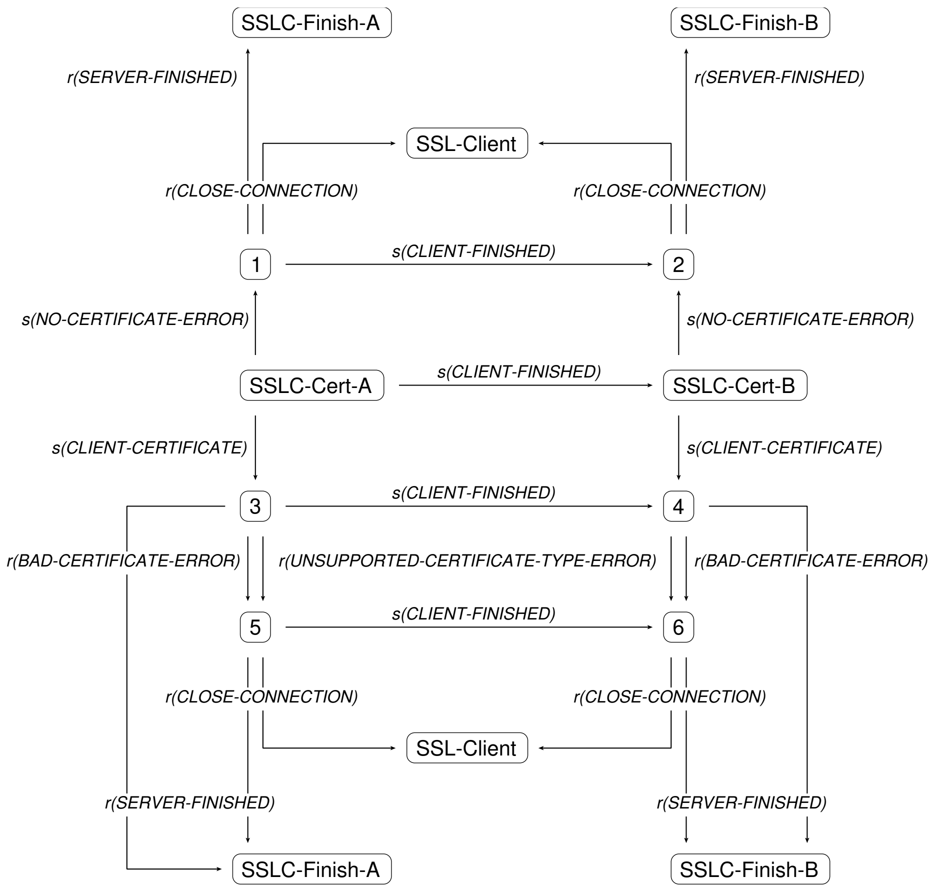

Figure 8: Client, if client certification requested

$$
\begin{aligned}
\text{SSLC-Cert-A} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLC-Cert-B} + \overline{\textsc{no-certificate-error}} \cdot \text{SSLCC1} + \\
&\quad\ \overline{\textsc{client-certificate}} \cdot \text{SSLCC3} \\
\text{SSLC-Cert-B} &\stackrel{\text{def}}{=} \overline{\textsc{no-certificate-error}} \cdot \text{SSLCC2} + \overline{\textsc{client-certificate}} \cdot \text{SSLCC4} \\
\text{SSLCC1} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLCC2} + \textsc{server-finished} \cdot \text{SSLC-Finish-A} + \\
&\quad\ \textsc{close-connection} \cdot \text{SSL-Client} \\
\text{SSLCC2} &\stackrel{\text{def}}{=} \textsc{server-finished} \cdot \text{SSLC-Finish-B} + \textsc{close-connection} \cdot \text{SSL-Client} \\
\text{SSLCC3} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLCC4} + \textsc{server-finished} \cdot \text{SSLC-Finish-A} + \\
&\quad\ (\textsc{bad-certificate-error} + \\
&\quad\ \textsc{unsupported-certificate-type-error}) \cdot \text{SSLCC5} \\
\text{SSLCC4} &\stackrel{\text{def}}{=} \textsc{server-finished} \cdot \text{SSLC-Finish-B} + (\textsc{bad-certificate-error} + \\
&\quad\ \textsc{unsupported-certificate-type-error}) \cdot \text{SSLCC6} \\
\text{SSLCC5} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLCC6} + \textsc{server-finished} \cdot \text{SSLC-Finish-A} + \\
&\quad\ \textsc{close-connection} \cdot \text{SSL-Client} \\
\text{SSLCC6} &\stackrel{\text{def}}{=} \textsc{server-finished} \cdot \text{SSLC-Finish-B} + \textsc{close-connection} \cdot \text{SSL-Client}
\end{aligned}
$$

4.5 Application Protocol Data Transfer and Termination

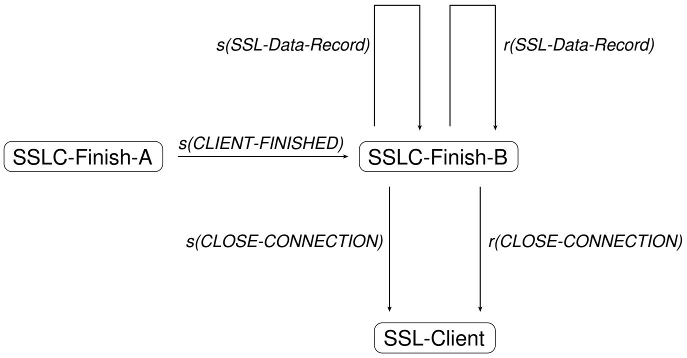

Figure 9: Client, terminate handshake and transfer data

$$
\begin{aligned}
\text{SSLC-Finish-A} &\stackrel{\text{def}}{=} \overline{\textsc{client-finished}} \cdot \text{SSLC-Finish-B} \\
\text{SSLC-Finish-B} &\stackrel{\text{def}}{=} (\overline{\textsc{ssl-data-record}} + \textsc{ssl-data-record}) \cdot \text{SSLC-Finish-B} + \\
&\quad\; (\overline{\textsc{close-connection}} + \textsc{close-connection}) \cdot \text{SSL-Client}
\end{aligned}
$$

5 Server-Side Handshake Analysis

There is no need to derive the server algebra from a new set of finite state diagrams, since the FSD's are topologically identical to those of the client. Deriving a generic algebra of a server from that of a client is a simple matter. All input messages become output messages, all output messages become input messages and the state labels require changing. The results of this transformation appear below.

5.1 Initial Server Negotiation Phase

$$
\begin{aligned}
\text{SSL-Server} &\stackrel{\text{def}}{=} \textsc{client-hello}(\textsc{no-sid}) \cdot \text{SSLS1} + \textsc{client-hello}(\textsc{sid}) \cdot \text{SSLS2} \\
\text{SSLS1} &\stackrel{\text{def}}{=} \overline{\textsc{server-hello}(\textsc{no-hit})} \cdot \text{SSLS-No-SID} + \overline{\textsc{no-certificate-error}} \cdot \text{SSLS3} \\
\text{SSLS2} &\stackrel{\text{def}}{=} \overline{\textsc{server-hello}(\textsc{no-hit})} \cdot \text{SSLS-No-SID} + \overline{\textsc{server-hello}(\textsc{hit})} \cdot \text{SSLS-Mid} \\
\text{SSLS3} &\stackrel{\text{def}}{=} \overline{\textsc{close-connection}} \cdot \text{SSL-Server}
\end{aligned}
$$

5.2 Master Key Negotiation

$$
\begin{aligned}
\text{SSLS-No-SID} &\stackrel{\text{def}}{=} (\textsc{bad-certificate-error} + \textsc{unsupported-certificate-type-error} + \\
&\quad\; \textsc{no-cipher-error}) \cdot \text{SSLSNS1} + \textsc{client-master-key} \cdot \text{SSLS-Mid} \\
\text{SSLSNS1} &\stackrel{\text{def}}{=} \textsc{close-connection} \cdot \text{SSL-Server}
\end{aligned}
$$

5.3 Middle Phase Server Negotiation

$$
\begin{aligned}
\text{SSLS-Mid} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLSM1} + \overline{\textsc{server-verify}} \cdot \text{SSLSM2} \\
\text{SSLSM1} &\stackrel{\text{def}}{=} \overline{\textsc{server-verify}} \cdot \text{SSLSM3} \\
\text{SSLSM2} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLSM3} + \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-A} + \\
&\quad\; \overline{\textsc{server-request-certificate}} \cdot \text{SSLS-Cert-A} \\
\text{SSLSM3} &\stackrel{\text{def}}{=} \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-B} + \\
&\quad\; \overline{\textsc{server-request-certificate}} \cdot \text{SSLS-Cert-B}
\end{aligned}
$$

5.4 Client Certification

$$
\begin{aligned}
\text{SSLS-Cert-A} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLS-Cert-B} + \textsc{no-certificate-error} \cdot \text{SSLSC1} + \\
&\quad\; \textsc{client-certificate} \cdot \text{SSLSC3} \\
\text{SSLS-Cert-B} &\stackrel{\text{def}}{=} \textsc{no-certificate-error} \cdot \text{SSLSC2} + \textsc{client-certificate} \cdot \text{SSLSC4} \\
\text{SSLSC1} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLSC2} + \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-A} + \\
&\quad\; \overline{\textsc{close-connection}} \cdot \text{SSL-Server} \\
\text{SSLSC2} &\stackrel{\text{def}}{=} \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-B} + \overline{\textsc{close-connection}} \cdot \text{SSL-Server} \\
\text{SSLSC3} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLSC4} + \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-A} + \\
&\quad\; (\overline{\textsc{bad-certificate-error}} + \\
&\quad\; \overline{\textsc{unsupported-certificate-type-error}}) \cdot \text{SSLSC5} \\
\text{SSLSC4} &\stackrel{\text{def}}{=} \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-B} + (\overline{\textsc{bad-certificate-error}} + \\
&\quad\; \overline{\textsc{unsupported-certificate-type-error}}) \cdot \text{SSLSC6} \\
\text{SSLSC5} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLSC6} + \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-A} + \\
&\quad\; \overline{\textsc{close-connection}} \cdot \text{SSL-Server} \\
\text{SSLSC6} &\stackrel{\text{def}}{=} \overline{\textsc{server-finished}} \cdot \text{SSLS-Finish-B} + \overline{\textsc{close-connection}} \cdot \text{SSL-Server}
\end{aligned}
$$

5.5 Application Protocol Data Transfer and Termination

$$
\begin{aligned}
\text{SSLS-Finish-A} &\stackrel{\text{def}}{=} \textsc{client-finished} \cdot \text{SSLS-Finish-B} \\
\text{SSLS-Finish-B} &\stackrel{\text{def}}{=} (\textsc{ssl-data-record} + \overline{\textsc{ssl-data-record}}) \cdot \text{SSLS-Finish-B} + \\
&\quad\; (\textsc{close-connection} + \overline{\textsc{close-connection}}) \cdot \text{SSL-Server}
\end{aligned}
$$

6 Recommendations

6.1 Forcing renegotiation of session keys

While SSL is obviously suited to layering between HTTP and TCP, which features many short data bursts, problems may arise with periods of uninterrupted data transmission. These may occur if SSL is used to encrypt a long TELNET session or if layered on a network protocol which features a single data burst. While most of the ciphers used by SSL are difficult to break by force, a substantial part of SSL's security lies in the fact that the encryption keys are changed very regularly. This is especially true given SSL's susceptibility to known-plaintext-attack while being used with any one key [DEN82, SCH94]. The more that data is transmitted under one key, the more it becomes worthwhile for a third person to try a brute force attack on the encryption system.

It is suggested that the *security escape bit*, outlined in the protocol definition should be used as a *protocol control character* to identify a new set of handshake messages in mid-session. Thus either the CLIENT-WRITE and CLIENT-READ keys could be recalculated or the master key could be renegotiated, even during a long transmission.

6.2 Additional error messages

The following error messages are missing from the current protocol document:

- UNSUPPORTED-AUTHENTICATION-TYPE-ERROR
- UNEXPECTED-MESSAGE-ERROR

The former is fatal for server certification and optionally fatal for client certification. The latter would just be useful in tidying connections when a client or server for some reason sends or receives a message in the wrong order. This would obviously be fatal for both parties and would probably indicate a problem with an implementation of the SSL protocol.

6.3 Clarification of client certification

BAD-CERTIFICATE-ERROR should be fatal for both client and server certification. It must be assumed that such an error has been generated by a false certificate (although noise could equally well have damaged the certificate). In such an instance the connection must be closed. If the connection is allowed to continue then it is possible that the server could be connected to a client with a forged identity. If this is not a problem for the server then it does not need to send the REQUEST-CERTIFICATE message in the first place. There seems little sense in using up bandwidth requesting a client certificate and then deciding that it does not matter if the certificate has been revoked, for instance.

6.4 Certificate Paths

In rfc1422 [KEN93], it states that a UA (User Agent – in this case SSL or the application using SSL) should be able to deliver a full certification path, on request – if not by default. This would allow for complete validation of client and server by either party if that were required. This is not currently implemented in the specification and it is recommended that an extra message REQUEST-CERTIFICATION-PATH be instituted. This would be a message that could be sent by either party if there is not enough information in the certification cache to completely verify the initial certificate. Thus it would be sent by the client immediately after a SERVER-HELLO message or by the server immediately after a CLIENT-CERTIFICATE message.

6.5 Client-requested certification

Currently the server can request certification of the client at any stage of a connection. Consideration should be given to allowing extra requested certification of the server – especially in the instance where the server is *pushing* important data at the client (as with the proposed Netscape Extensions to HTML). This would be in addition to the server authentication at the beginning of an SSL session.

6.6 Certification Authorities

RSADSI is the only commercial certification authority currently supported by Netscape however in its legal documentation it states that an RSA certificate may only be used with official RSA products. Given that most RSA products are subject to US export controls and are thus not available overseas. It does appear that development of a secure non-US Commerce server which can be accessed by Netscape Browsers has been, maybe unwittingly, unfairly restricted.

References

[DEN82] Dorothy Denning *Cryptography and Data Security*. ISBN 0-201-10150-5, Addison-Wesley, 1982

[HAL93] Fred Halsall *Data Communications, Computer Networks and Open Systems – 3rd Edition*. ISBN 0-201-56506, Addison-Wesley, 1993

[HIC95] Kipp Hickman *The SSL Protocol*. Netscape Communications Corporation, Feb 1994

[KEN93] S. Kent *Privacy Enhancement for Internet Electronic Mail – Part II*. RFC 1422, IAB IRTF PSRG, IETF PEM, Feb 1993

[SCH94] Bruce Schneier *Applied Cryptography: Protocols, Algorithms and Source Code in C*. ISBN 0-471-59756-2, John Wiley & Sons, 1994