MULTIPLE-CHANNEL SECURITY MODEL AND ITS IMPLEMENTATION OVER SSL

by

YONG SONG

B. Eng., Huazhong University of Science and Technology, China, 1994
M. Eng., South China University of Technology, China, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical and Computer Engineering)

We accept this thesis as conforming
to the required standard

_____
_____
_____
_____

THE UNIVERSITY OF BRITISH COLUMBIA
September 2004

Abstract

Multiple-Channel SSL (MC-SSL) is a new model and protocol to secure client-server communication. In contrast to SSL, which provides a single end-to-end secure channel, MC-SSL can provide applications with multiple channels, and each channel can have a various number of application proxies, its own cipher suite and data flow direction. MC-SSL enjoys a few advantages over SSL including end-to-end security and selective security.

This thesis first introduces the background and analyzes some security-related problems in client-server communications, especially when resource-constrained handheld devices are involved. The multiple-channel model of MC-SSL is then presented as a solution, and compared with other related work. This thesis further focuses on the protocol design and implementation of MC-SSL.

Table of Contents

Abstract ii
Table of Contents iii
List of Figures v
Acknowledgements vi
Chapter 1 Introduction 1
1.1 The Problem 2
1.2 The Solution 5
1.3 Chapter Organization 6
Chapter 2 Problem Motivation 7
2.1 Problem with Proxies 8
2.2 Limitation of Cipher Suites and Channel Direction 10
2.3 Problem with Negotiation 12
Chapter 3 MC-SSL Model 13
3.1 Multiple-Channel Model 13
3.2 Notes on the MC-SSL Model 17
Chapter 4 Related Work 22
4.1 ITLS 22
4.2 Independent SSL Connections 23
4.3 Changing Cipher Suite in SSL 25
4.4 Data Compression 26
4.5 An SSL Extension for a Cleartext Channel 27
4.6 Acceleration of Cryptographic Operations 28
4.7 XML-based Solutions 29
4.8 Security Analysis and Development of SSL 31
Chapter 5 Protocol Design 34
5.1 Protocol Architecture 34
5.2 Initial Handshake Protocol 35
5.3 Proxy Channel Protocol 37

5.4 Application Data Protocol 44
5.5 Alert Protocol 47
5.6 Secondary Channel Protocol 47
5.7 Restriction on Channel Directions 53
5.8 Channel Cancellation 54
5.9 Abbreviated Handshake 54
5.10 Discussion 56
Chapter 6 Implementation and Case Studies 61
6.1 Message Formats of MC-SSL 61
6.2 State Diagrams 61
6.3 System Diagram of the Prototype 64
6.4 Protocol Names 65
6.5 An Application Case Study 66
Chapter 7 Conclusion and Future Work 73
Bibliography 75
Appendix A Overview of SSL 80
Appendix B Message Formats of MC-SSL Protocol 83

List of Figures

Figure 2-1: The end-to-end model and proxy chain model of SSL 8
Figure 3-1: Multiple cipher suites inside a Connection 14
Figure 3-2: Proxy channel model of MC-SSL 14
Figure 3-3: Multiple-channel model of MC-SSL 15
Figure 3-4: A sample of WAP 2.0 architectures [5] 18
Figure 3-5: Four Methods to Complete a Task 20
Figure 3-6: Reducing the Hop Number of a Proxy Channel 21
Figure 4-1: Independent SSL Connections 24
Figure 4-2: The Roadmap of Web Services Security [23] 30
Figure 5-1: Two-layer Architecture of MC-SSL 35
Figure 5-2: Initial Handshake Protocol 36
Figure 5-3: Primary Proxy Channel Protocol 38
Figure 5-4: The enhanced authentication: (a) first stage, (b) second stage 43
Figure 5-5: Application Data Protocol 45
Figure 5-6: Negotiating Secondary Channels 48
Figure 5-7: Adding Channel Id in an SSL Packet 50
Figure 5-8: Handshake Protocol to Switch Cipher Suites 51
Figure 5-9: Implementation of Channel Directions 53
Figure 5-10: Channel Cancellation Protocol 54
Figure 6-1: Basic State Diagram at a Client or Server 63
Figure 6-2: Basic State Diagram at a Proxy 64
Figure 6-3: System Diagram of the Prototype 65
Figure 6-4: Channel Planning for Online Banking 68
Figure A-1: SSL Architecture [40] 80
Figure A-2 (a): Full Handshake of SSL 82
Figure A-2 (b): Abbreviated Handshake of SSL 82

Acknowledgements

I am very grateful to Prof. Victor C.M. Leung and Prof. Konstantin Beznosov for giving me extraordinary guidance, support, and encouragement. Because of their superb supervision, I can choose a very interesting topic, carry out the research, publish two papers, write and defend my thesis in a timely and smooth way. I particularly appreciate that Prof. Leung supported me to do research related to security and mobile applications, that Prof. Beznosov helped me publish research results, and that they spent a lot of time to review my reports, papers, and thesis.

This work was funded by grants from Telus Mobility and the Advanced Systems Institute of BC, and by the Canadian Natural Sciences and Engineering Research Council under grant CRD247855-01.

Chapter 1

Introduction

With the Internet developing into a global and open information infrastructure, information security has been increasingly recognized as an essential aspect of information technologies. Meanwhile, Internet is extending from wireline to wireless networks. There are a growing number of handheld devices such as cellular phones, PDAs and palmtops, which are able to access Internet applications such as Web, email, multimedia, etc. How to secure client-server communication between a handheld device and an Internet server has been a challenge. This thesis presents a new security model and protocol named Multiple-Channel SSL (MC-SSL) to secure client-server applications. MC-SSL protocol takes an approach of extending SSL/TLS (Secure Socket Layer/Transport Layer Security) [1] to meet special requirements of handheld devices. SSL/TLS is the *de facto* security protocol for Web, email, and some other popular Internet applications. Although the problem of handheld devices is the focus of MC-SSL, MC-SSL is designed to be a general security protocol for various client-server communication scenarios, not just for handheld or mobile devices. Reusing SSL/TLS in MC-SSL is consistent with this goal.

A handheld device has much more constraints than an ordinary computer in terms of power, processor, memory, display, and so on. The access interfaces of handheld devices range from 2G/2.5G/3G cellular networks, Wireless LAN, Bluetooth, to dial-up and LAN. Some of them are slow, unreliable, and/or expensive. Note that a handheld device

is still resource-constrained even though it uses a wireline interface such as LAN for communication. Beside, the operating system and software on a handheld device often have fewer functions than those on an ordinary computer. However, many applications and protocols in the Internet are designed mainly for ordinary computers. For these reasons, it is a challenge to secure client-server communications for handheld devices.

1.1 The Problem

The first problem is about security risks of application proxies. Due to their constraints, many handheld devices have to use application proxies to help them access servers in the Internet. For example, a Web site is likely not designed for a handheld device, and the micro browser cannot correctly display Web pages if they are not transformed either by the Web server or by an intermediary proxy. Unfortunately, most Web sites still do not provide Web pages specifically designed for handheld devices. Web sites protected by SSL/TLS have the same problem. After all, it is not easy to design, test, and maintain Web pages for every type of handheld device. The development of technology has been proliferating handheld devices with various capabilities. In addition, even ordinary computers sometimes need an application proxy. The requirement of an application proxy is not a problem by itself, but if sensitive data or content is in transit, it becomes difficult to achieve end-to-end security between a client and a server. For example, we cannot simply employ end-to-end encryption on all data transported in a session to achieve end-to-end confidentiality because an application proxy requires decrypting some data in order to read and even modify some of them to provide proxy services such as transforming and filtering a Web page. The exposure of sensitive data to a proxy could

result in information leakage and tampering if a proxy is not trustworthy. In brief, we need to adopt, or invent if necessary, some techniques to simultaneously meet the functional requirement for application proxies and the security requirement for end-to-end security.

The second problem is about how to reduce redundant cryptographic protection in client-server communications. A well-known guideline of designing hardware and software for handheld devices is to reduce the consumption of battery power and processor time as much as possible. Cryptographic algorithms such as RSA (Rivest, Shamir, Adleman) [2], 3DES (Triple DES) [2], and AES (Advanced Encryption Standard) [2] are computationally expensive for handheld devices. If a processor is fully dedicated to security processing, the processing requirements for 3DES, AES, SHA (Secure Hash Algorithm) [2] and MD5 (Message Digest 5) [2] at 10 Mbps are 535.9, 206.3, 115.4 and 33.1 MIPS (Millions of Instructions Per Second), respectively [3]. In comparison, a handset processor such as Intel's StrongARM processor SA-1110 is capable of delivering around 235 MIPS at 206 MHz [3]. For a powerful Web or email server, cryptographic operations, together with other tasks, can overload the processor if there are many users accessing the server at the same time. In the past several years, many large websites such as Yahoo Mail, Hotmail have significantly changed their way of using SSL. They have managed to reduce the SSL-protected communication to the minimum. Currently Yahoo Mail does not provide SSL protection in the standard mode; even in the secure mode, only is the login process (i.e., user id and password) protected by SSL (www.yahoo.com). Similarly, Hotmail (www.hotmail.com) protects only the login process. These facts demonstrate that cryptographic protection, such as SSL,

consumes a lot of processor time, and it is important to avoid redundant protection. Here is a typical example of redundant protection: a lot of information in a Web page only requires integrity or authenticity, but they are encrypted using a 128-bit cipher in an HTTPS (HTTP over SSL) session. In fact, different user and service providers may have different opinions about security and privacy. Their requirements should be flexibly taken into account. In summary, there is a need to facilitate selective security. Although choosing or switching between HTTP and HTTPS URL links can provide selective security to some degree, it works only for Web applications at coarse granularity [4]. Applications require fine granularity of selective security.

SSL and its variants, TLS and WTLS, are the *de facto* application security protocols in the Internet. Note that SSL/TLS/WTLS are referred as SSL in some places of this thesis. An ordinary desktop usually uses SSL/TLS to secure its communication with a Web server or an email server. A cellular phone can use WTLS (Wireless Transport Layer Security), a variant of TLS, to secure its communication with a WAP (Wireless Application Protocol) gateway [5]. Some high-end PDAs are able to directly access Web sites without a proxy, and they support SSL. Palm™ Tungsten™ C and Dell™ Axim™ X30 are two examples. Despite its popularity, SSL alone cannot solve the above problems very well: SSL is not able to enhance the end-to-end security in the presence of application proxies, and it is not able to sufficiently support selective security either. Chapter 2 will further discuss the limitations and drawbacks of SSL. In light of these observations, this thesis project investigates the above problems, and tries to find a solution to them.

1.2 The Solution

By delving into the problem from different aspects such as the requirements of handheld devices, the capabilities of client and server devices, and the limitations of SSL, this thesis proposes Multiple-Channel SSL (MC-SSL). As its name indicates, MC-SSL is a protocol that consists of multiple channels between a client and a server. Each of them could be an end-to-end channel without any proxy, or a proxy channel with some intermediary proxies, and each of them has its own cipher suite and data flow direction. During a particular session, a client and a server can set up some channels that they need by negotiation, and selectively use them to transport data and obtain proxy services. The design objective of MC-SSL is to provide several kinds of channels, and let client and server themselves decide what channels are needed and how to use them. This idea can be realized in different ways, but we have chosen the approach of extending SSL instead of proposing an entirely new protocol. A new protocol layer that is in charge of channel control is added on top of SSL. The approach of reusing SSL minimizes the effort that is required to deploy MC-SSL, and facilitates the convergence of wireless networks and wireline networks.

In comparison with SSL, MC-SSL enjoys several advantages. First, MC-SSL significantly enhances end-to-end security in the presence of application proxies. Second, MC-SSL supports multiple cipher suites so that client and server can use different cipher suites to transport data with different security requirements. Third, MC-SSL supports restriction on channel directions. A simplex MC-SSL channel can prevent a response channel from being turned into a request channel by a person-in-the-middle, and vice versa. Finally, MC-SSL supports channel negotiation based on security policies, device

capabilities, and security attributes of data. Consequently, MC-SSL can better fulfill the diverse requirements brought up by different client and server devices, applications, and users.

MC-SSL has its downside as well. First, MC-SSL protocol is not as simple as SSL in terms of providing data security. Client and server must wisely define their security policy about what channels are required and how to use them. Second, sometimes MC-SSL alone cannot satisfy complex security-related requirements of client-server applications. Other techniques, such as XML and XML Security, could be necessary. Third, it is hard to eliminate security risks of an application proxy. Application developers and users must choose and use proxies discreetly.

The contribution of this thesis includes the following: the modeling and protocol design of MC-SSL, the research on selective security and proxy-related end-to-end security, the prototype implementation and case studies of MC-SSL. Main research results have been published in two papers [6, 7].

1.3 Chapter Organization

The rest of this thesis is organized as follows. Chapter 2 further discusses the problems and analyzes the limitations of SSL. Chapter 3 presents the high-level model of MC-SSL. Chapter 4 studies relate work. Chapter 5 discusses the protocol design of MC-SSL. Chapter 6 describes the prototype implementation, and demonstrates an application case of MC-SSL. Chapter 7 concludes the thesis. In addition, Appendix A gives a brief overview of SSL, and Appendix B defines the message formats of MC-SSL protocol.

Chapter 2

Problem Motivation

Although SSL is the *de facto* application security protocol in the Internet, it has several limitations. First, SSL cannot help applications prevent information leakage and impersonation at an application proxy when an application proxy is involved. Second, SSL does not support multiple cipher suites inside a connection. Third, SSL does not support simplex channels, in which data is transmitted only in one direction. Fourth, SSL does not have sufficient negotiation mechanisms to support selective security and improve end-to-end security.

A simplified communication model of SSL is illustrated in Figure 2-1. In the upper part of Figure 2-1, SSL provides client **C** and server **S** with a point-to-point secure channel over a TCP connection. The security of the channel is achieved by making use of a cipher suite, which consists of a key exchange algorithm, a bulk encryption algorithm, and a hash algorithm. A sample cipher suite of SSL is TLS_RSA_WITH_IDEA_CBC_SHA [1]. This cipher suite uses RSA algorithm [2, 8] to perform authentication and key exchange, IDEA (International Data Encryption Algorithm) [2, 8] to perform encryption and decryption, and SHA-1 (Secure Hash Algorithm) [2, 8] to generate MAC (Message Authentication Code) [1]. MAC is for protecting data integrity. CBC (Cipher Block Chaining) [2, 8] is a mode of operation for block ciphers such as IDEA. Appendix A gives an overview of TLS. Please refer to RFC 2246 [1] for the details of SSL/TLS.

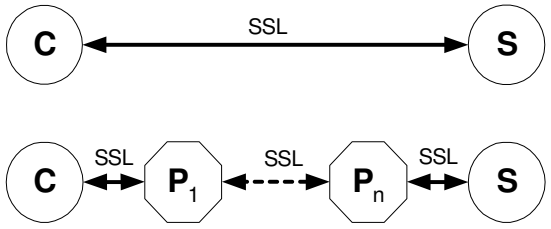

Figure 2-1: The end-to-end model and proxy chain model of SSL

2.1 Problem with Proxies

An application proxy (or gateway) is a proxy that works at the application layer, which means that it must read and even modify some application data or messages. The purpose of an application proxy could be virus scanning, content transformation, or compression. Application proxies are referred briefly as proxies sometimes in this thesis. Handheld devices tend to need proxies because they do not have enough resources, such as processor, memory, power, and software, to perform some tasks, or because the access network in use is slow and/or unreliable, such as a GSM/GPRS (Global System for Mobile Communications / General Packet Radio Service) network [9]. WAP [5] 1.X gateway is such a proxy that is currently used by many cellular phones to access Internet. Another example is T-Mobile Internet Accelerator [10], which accelerates the access speed of a PDA by compressing images and text in Web pages. Similarly, a Web clipping proxy server [11] helps Palm PDAs access Web sites by transforming Web pages into suitable forms. Recently the first worm against cellular phones has appeared [12]. Presumably, more and more viruses or malware against handheld devices will emerge in the future; however, it is hard to have full-fledged anti-virus software installed in some handheld devices. An anti-virus proxy is a possible solution to protect them from viruses in Web pages, downloaded software, email

attachments, and so on. Proxies are useful for desktops as well. For instance, some companies deploy application firewalls to protect their intranet from computer viruses. These application firewalls are actually proxies that scan Web pages and emails, and filter out questionable content.

While SSL can provide a secure point-to-point channel, it has problems in the presence of application proxies: if an application proxy **P** is involved, client **C** normally sets up an SSL connection with **P**, and **P** acts as the delegate of **C** and sets up another SSL connection with server **S**. The proxy chain model of SSL is illustrated in the lower part of Figure 2-1, in which there is a number of application proxies between **C** and **S**. In this model, SSL can protect the communication between any two neighbouring entities, but cannot prevent a proxy in the chain from reading and modifying sensitive data. This model assumes unconditional trust in all proxies at least by one endpoint, **C** or **S**. This can be satisfied only if proxies are administrated by the organization or individual that also administrates **C** or **S**. In other cases, **C** or **S** has to take the risks of information leakage and tampering unless the exchanged information is non-sensitive. In other words, a proxy could compromise the end-to-end security between **C** and **S**. Note that if a proxy works below the application layer, for example, at transport layer, then **C** and **S** can still establish an end-to-end SSL session. For this reason, SOCKS and NAT (Network Address Translation) do not affect the normal operation of SSL. The proxy chain model of SSL is only related to application proxies.

A typical example of SSL chain proxy model is the WAP 1.X gateway architecture that is shown in Figure 2-2. The communication security between a WAP device and a WAP gateway is protected by WTLS, a variant of SSL protocol. Obviously, the WAP 1.X gateway shown in Figure 2-2 is an application proxy because it performs content transformation,

recoding, and/or compression for the content carried by HTTP or WSP/WTP protocols. Since this architecture is a proxy architecture that employs SSL, it has the same problem that an SSL proxy chain model has. The architecture is secure only when the gateway is trustworthy. For instance, the gateway is provided by the owner of the Web server.

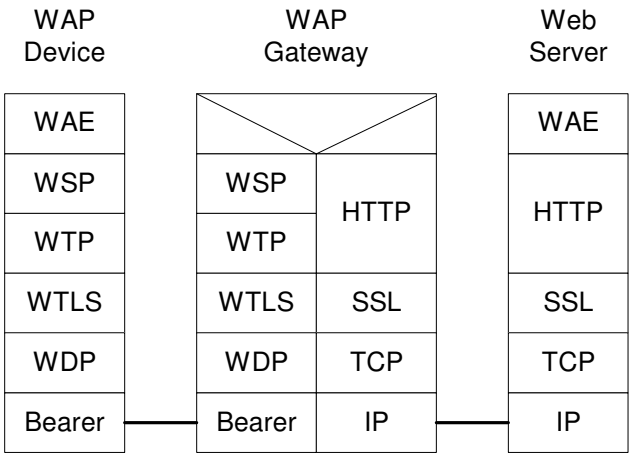

Figure 2-2: WAP 1.X gateway architecture [5]

2.2 Limitation of Cipher Suites and Channel Direction

The second limitation of SSL is that it can use only one cipher suite at any time during a session. In fact, SSL allows changing cipher suite during a session by means of re-negotiation, namely doing a full handshake, which is shown in Figure A-2 (a); however, re-negotiations are inefficient because of the message interaction and the certificate verification process in a full handshake has high communication cost, especially for a wireless handheld device. Another limitation of SSL is that it cannot support simplex channels with a direction that restricts the data flow. SSL only provides a single duplex channel, in which the cipher suites for both directions are identical. When requests and responses need different protection, SSL cannot use different cipher suite for them because it is inefficient to change cipher suite in every round of client-server interaction. For these and other reasons, few

applications change their cipher suites in an SSL session. As a result, many data are overly protected with redundant cryptographic operations. As quoted in Chapter 1, the processing requirements for 3DES, AES, SHA-1 and MD5 at 10 Mbps are 535.9, 206.3, 115.4 and 33.1 MIPS (Millions of Instructions Per Second), respectively [3]. 3DES and AES are two popular algorithms for encryption/decryption, and SHA-1 and MD5 are two popular hash algorithms. The fact that many large Web sites try to minimize the traffic volume protected by SSL also manifest the high cost of cryptographic operations.

There are typically several types of redundant protection. First, a lot of information is not confidential but it is still encrypted together with confidential information using the same strong cipher in an SSL session. For example, a Web page from a bank's Web site carries account numbers and their balances, which are usually considered confidential; however, most bytes in the Web page, including other texts, images, HTML tags, scripts, Java applet, and so on, are not confidential. For those bytes, HMAC (keyed-Hash Message Authentication Code) based on MD5 or SHA-1 is usually sufficient in order to ensure data integrity. The processing requirements of MD5 are only 33.1 MIPS, but AES plus MD5 is 206.3 MIPS [3]. The latter is over 6 times as slow as the former. Second, some information is already secured at application layer. For example, software, emails, and documents could have been digitally signed or encrypted with XML Security, Web Service Security, PGP, or other techniques. Extra protection by SSL is likely redundant. Third, some services or applications only require authentication. They use SSL to do client and server authentication. However, after login stages are passed, they do not require further data protection although privacy is still a concern to some degree. The services of Yahoo Mail and Hotmail can be categorized into this type. Some Web sites use proprietary techniques based on cookies,

hidden fields, or URL parameters to carry authentication and session information after authentication, which could result in session hijacking and information leakage if those techniques are not carefully designed. Non-Web applications have similar issues.

To summarize, the requirement for communication security does not entail excessive data protection; moreover, security is tightly related to other requirements. For SSL, supporting one cipher suite at a time combined with the relatively high cost of changing cipher suites just in time makes it difficult for applications to optimize the strength of data protection according to the changes in the sensitivity of the data in the channel. Applications need a security protocol to support selective security at fine granularity.

2.3 Problem with Negotiation

To decide whether or not and how to use proxies, multiple cipher suites, and simplex channels, **C** and **S** must exchange sufficient information in order to make right decisions that optimize the combination of different channels. Generally, **C** needs to inform **S** of its device capabilities and security policy. For example, **C** may define whether proxies are allowed to process data with sensitivity below a certain level, what cipher suites are strong enough to protect data with a certain level of sensitivity, and so on. Lack of negotiation support is the third limitation of SSL. Moreover, the core of these limitations is that the negotiation and decision process of SSL does not take the security policies, device capabilities, and other important factors into account. These functional limitations form a mismatch gap between SSL and the diverse requirements of client-server applications. When handheld devices and mobile applications become more popular, the gap will likely become more apparent.

Chapter 3

MC-SSL Model

The intent of MC-SSL is to overcome the limitations of SSL, and narrow the gap between SSL and the requirements of applications. To address the first limitation (the dilemma between unconditionally trusted proxies and no proxy at all), we introduce proxy channels to support partially trusted proxies. To address the second limitation that only one cipher suite at a time and for both directions, MC-SSL supports multiple cipher suites and channel direction. MC-SSL supports channel negotiation according to the parties' security policies, device capabilities, and security attributes of data in order to address the third limitation of SSL.

3.1 Multiple-Channel Model

In SSL, a channel is associated with a cipher suite, which consists of a key exchange algorithm, a cipher, and a hash algorithm, e.g., {RSA, 3DES_EDE_CBC/168, SHA-1}. The hash algorithm is used to compute Message Authentication Code (MAC). In MC-SSL, a cipher suite consists of only two elements: a cipher for data encryption and decryption, and a hash algorithm for MAC, and hence it can be denoted as follows:

{cipher and key size, hash algorithm for MAC} (1)

An MC-SSL connection can have multiple cipher suites. We can characterize a point-to-point connection as follows: {*point 1, point 2, key exchange algorithm,* {*cipher suite 1, cipher suite 2,* …}}, where each cipher suite forms a channel. Note that the key exchange algorithm no

longer belongs to a cipher suite, but becomes an attribute of a connection. Every MC-SSL connection must first negotiate a strong cipher suite (e.g. a 128-bit cipher plus SHA) to form the primary channel, the backbone for setting up and controlling other channels in the same connection. A primary channel is the first channel established over an SSL connection, and it can be set up with the unchanged SSL protocol. Other channels established over an SSL connection are referred as secondary channels. They are new channels added to an SSL connection to support multiple cipher suites over an SSL connection. The sample connection shown in Figure 3-1 can be characterized as {**A**, **B**, RSA, {CS1, CS2, CS3, CS4}}, where RSA is the key exchange algorithm, and CS1 through CS4 are cipher suites for channels 1 to 4. Among them, channel 1 is the primary channel.

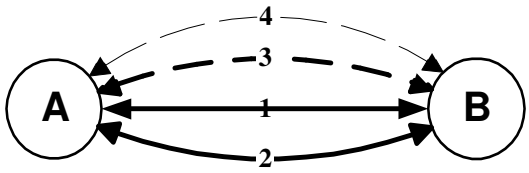

Figure 3-1: Multiple cipher suites inside a Connection

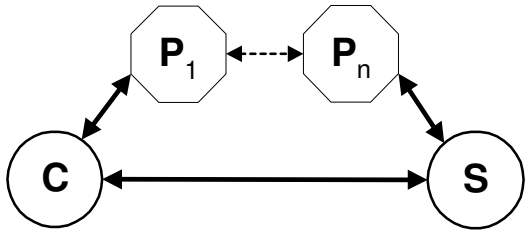

Figure 3-2: Proxy channel model of MC-SSL

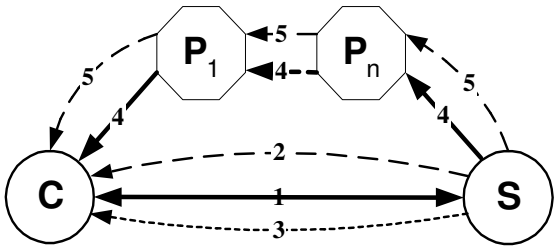

Figure 3-3: Multiple-channel model of MC-SSL

Figure 3-2 shows the proxy model of MC-SSL, in which point-to-point SSL connections collectively form a shape of an arc. **C**-**S** is an end-to-end channel, and **C**-**P**$_1$- … -**P**$_n$-**S** is a proxy channel. In this model, **C**-**P**$_1$- … -**P**$_n$-**S** is a channel that relies on the **C**-**S** channel to do channel negotiation and application data transportation. The **C**-**S** channel must exist before the negotiation of the proxy channel is started. Through the **C**-**S** channel, **C** and **S** exchange messages about what proxies they want and other parameters of the proxy channel. After that, **C** and **S** interact with proxies to set up the proxy channel. The **C**-**S** channel is also used to control data transmission through the proxy channel. **C** or **S** can deliberately choose one of the two channels to transport data according to the security requirements of the data. For example, sensitive information, such as passwords and credit card numbers can be transported using the end-to-end channel. An MC-SSL session can have zero or more proxy channels. Each of them and the corresponding end-to-end channel reflect the proxy model illustrated in Figure 3-2.

The combination of the proxy model and the multiple cipher suites is the multiple-channel model illustrated in Figure 3-3. In MC-SSL, a channel is a communication “pipe” with a certain cipher suite and a various number of application proxies. If there is no application proxy in the channel, then it is an end-to-end channel; if there is no cipher suite

for the channel (the cipher suite is null), then it is a plaintext channel. In addition, a channel can be duplex, simplex, or inactive. The restriction on channel direction only applies to application data message. An MC-SSL channel can be characterized by the following set of attributes:

$$\text{channel} \equiv \{ID, E_1, E_2, CS, \{P_1, P_2, \ldots, P_n\}, D\} \quad (2)$$

ID is the identity number of a channel. E_1 and E_2 are either DNS names or IP addresses of the corresponding endpoints. Cipher suite, *CS*, is defined by expression (1). A proxy (P_i) is identified by its DNS name or IP address. A channel can have zero or more proxies. Direction, *D*, indicates whether a channel is a duplex channel, an inactive channel, or a simplex channel pointing to one of the two endpoints. An inactive channel cannot be used to transport any application data. The sample MC-SSL session in Figure 3-3 has five channels. Among them, channel 1 and 4 are primary channels, and others are secondary channels. Note that an MC-SSL session can have multiple primary channels. The number of primary channels in an MC-SSL session is equal to the number of SSL connections that have **S** as an endpoint. Channel 2, 3, and 4 are negotiated through channel 1, and channel 5 is negotiated through channel 4. In addition, only channel 1 is a duplex channel for application data; others are simplex channels from **S** to **C**. In this application scenario, **C** uses channel 1 to send encrypted requests to **S**, and **S** chooses one of the five channels to send back responses.

The protocol developed from the above model is described in Chapter 5. It is designed and implemented over SSL. SSL is used to negotiate primary channels in MC-SSL, and is extended to support secondary channels. However, the multiple-channel model of MC-SSL and its channel definition are separated from SSL. In other words, the model of MC-SSL is

not bound to SSL and the transport layer. For example, one can develop an entirely new protocol from the model using XML Security [13, 14] at application layer.

3.2 Notes on the MC-SSL Model

An MC-SSL session always starts with setting up the primary end-to-end channel, namely channel 1 in Figure 3.3. This channel is the first channel and the basis to negotiate other channels in an MC-SSL session. In other words, MC-SSL is based on end-to-end negotiation through SSL. Section 3.2.1 and 3.2.2 discusses the feasibility and the flexibility of end-to-end negotiation.

A proxy channel could be a single-hop or multi-hop proxy channel. Section 3.2.3 discusses the preference for single-hop proxy channels, and a method to achieve this purpose.

3.2.1 Feasibility of End-to-end Negotiation

The first issue is that MC-SSL requires both client and server to support MC-SSL. One important difference between SSL proxy chain and MC-SSL proxy channel is that MC-SSL must establish the end-to-end channel at first; therefore, both client and server must support MC-SSL, which in turn requires them to support SSL since MC-SSL protocol is based on reusing SSL. It is not a problem for ordinary computers, but is it a problem for handheld devices? Actually, if a device is able to support SSL, it should be able to support MC-SSL. In comparison, client and server in the SSL proxy chain model can have very different protocol stacks. For example, a WAP device supports WTLS while an Internet server supports SSL, but they can communicate by means of a WAP gateway.

The device capability to support MC-SSL should not be a challenge for most handheld devices in the future. Even for WAP devices such as cellular phones, WAP 2.0 specification [5] has introduced new architecture and protocol stack that supports SSL. Figure 3-4 shows an example of WAP 2.0 architecture, in which a WAP device communicates with a Web server using TLS. New protocols such as Wireless Profiled TCP (denoted as TCP* in Figure 3-4), together with new development (e.g. 3G) of cellular networks and wireless terminals, make SSL/TLS feasible in cellular networks. Apart from cellular networks, some high-end PDAs that access Internet with WLAN, Bluetooth, or dial-up are already able to support SSL, as mentioned earlier in Chapter 1.

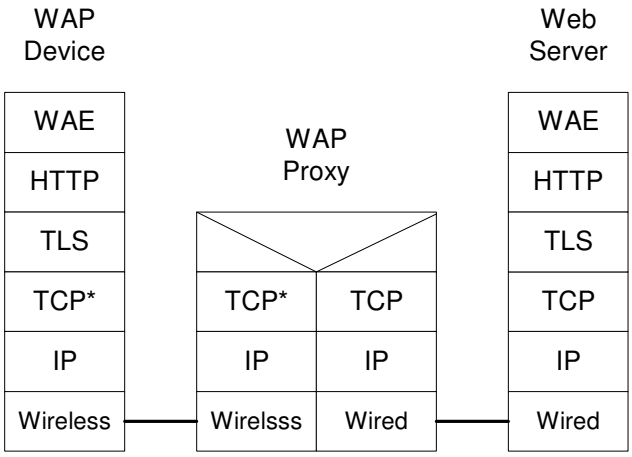

Figure 3-4: A sample of WAP 2.0 architectures [5]

3.2.2 Flexibility of End-to-end Negotiation

The approach of negotiating proxies through the end-to-end channel is more flexible than the traditional usage of a proxy. Traditionally, a client alone decides which proxy it should use. The parameters of a proxy are configured by a user in the client-side software. In comparison, the approach of MC-SSL is more flexible and securer than the traditional way

although the downside is that end-to-end negotiation requires both client and server to support MC-SSL. This section explains why the end-to-end negotiation in MC-SSL is more flexible. The security issues of proxy channels are discussed in Section 5.9.1 after the protocols of MC-SSL are presented.

Figures 3-5 shows four different methods to process data and send it to **C**. The first method uses a proxy channel of MC-SSL. **S** first delivers data to **P**, then **P** forwards it to **C** after processing, and finally a message that controls the processing is sent to **C** through the end-to-end channel. The Application Data Protocol of MC-SSL is presented in Section 5.4. Using the same proxy server, we can use the second and the third method to accomplish the same task, but **P** is not a proxy any more. The second method is generally better than the third one for the following reasons: processed data can be stored at S to serve other clients, and a "thin client" is generally better than a "fat client", especially when **C** is a handheld device connected with a low-speed or traffic-billing access network. If **S** or **C** can perform **P**'s functions, the fourth method, which is the simplest one among four methods, can be used. The point is that without negotiation **C** and **S** cannot automatically decide which method they should use. Moreover, choosing proxies or servers could be more complex than this simple case. Traditional way of using a proxy requires that a user knows beforehand what proxy is needed. MC-SSL is more flexible because choosing proxies is based on bilateral negotiation between client and server.

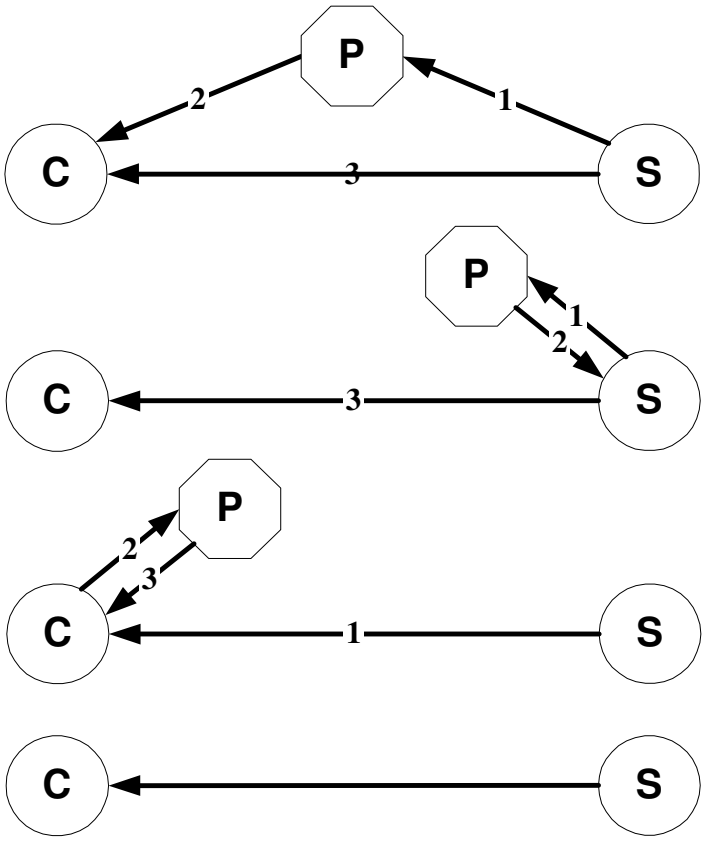

Figure 3-5: Four Methods to Complete a Task

Several other factors affect the choice of proxies. For example, a proxy could be hosted by the owner of the server, the client user (e.g. on home PC), the client user's company, a telecommunication company, the manufacture or seller of the client device, government, an industry association (e.g. a banking association), an independent proxy service provider, and so on. They may have different levels of trust, functions, and performance. MC-SSL allows client and server to agree on the choice of proxies.

3.2.3 The Preference for Single-hop Proxy Channels

The proxy channel model of MC-SSL does not limit the number of proxies in a proxy channel; however, it is easy to realise that the fewer proxies are involved, the securer and the

more efficient the channel is. The delays and security risks increases with the number of proxies. If possible, one might as well use the simple technique shown in Figure 3-6 to reduce the number of "hops" in a proxy channel. In the figure, **P1** acts as the "cluster head", the representative of the proxy "cluster" that consists of **P1**, **P2**, and **P3**, so that the proxy channel is transformed into a single-hop proxy channel.

It is not always possible to transform all proxies into a single hop, for example, when two proxies belong to different administrative domains. Therefore, one still needs to consider how to support multi-hop proxy channels. Section 5.3 presents protocols for setting up a single-hop proxy channel or a multi-hop proxy channel.

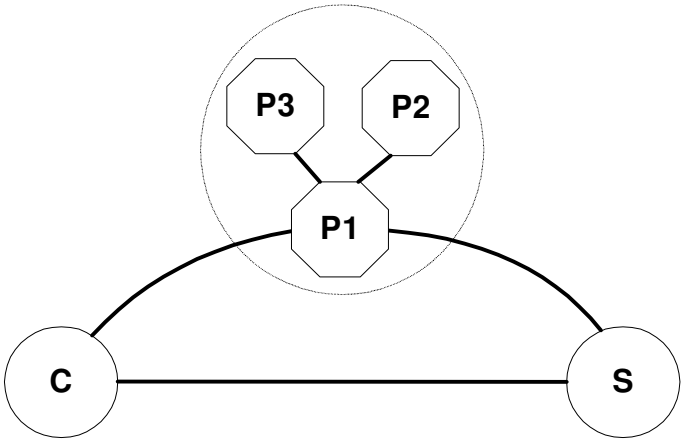

Figure 3-6: Reducing the Hop Number of a Proxy Channel

Chapter 4

Related Work

The limitations of SSL, and the problems of end-to-end security and selective security, have already been noticed by other researchers. There are related approaches to solving these problems. Some of them are tightly related to SSL/TLS/WTLS, while others are not. This chapter describes them, and compares them with MC-SSL.

4.1 ITLS

Integrated Transport Layer Security (ITLS) [15] is proposed by Kwon et al. to solve the end-to-end security problem of WAP, in which WTLS and SSL connections in tandem form an SSL chain model. The goal of ITLS is to prohibit the WAP gateway from having the plaintext message. In WTLS, **C** and **P** exchange and share a secret key, K_{CP}, and **P** and **S** exchange and share another secret key, K_{PS}. **C**-**P** and **P**-**S** are two independent SSL connections. In ITLS, **C** and **P** still exchange and share a secret key, K_{CP}, but **P** and **S** do not share a secret key any more. Instead, **C** and **S** exchange and share a secret key, K_{CS}. ITLS does not need any change of SSL/TLS at **S**, but has to slightly modify WTLS at **C** and **P**. In ITLS, **C** encrypts a message twice for **S** and **P** using K_{CS} and K_{CP} in the order named. **P** decrypts the cipher text using K_{CP} and then sends it to **S**. In reverse, **S** encrypts a message using K_{CS}, and sends it to **P**. **P** encrypts it again using K_{CP}, and then forwards it to **C**. **C** decrypts twice using K_{CP} and K_{CS} to get the message. ITLS can achieve end-to-end security for WAP 1.X devices without any change at the server side.

However, ITLS has several disadvantages if compared with MC-SSL. First, ITLS only improves WTLS, which is only used in WAP 1.X architecture and devices. WAP 1.X will be likely obsolete in the near future. As described in Section 3.2.1, WAP 2.0 has already supported SSL/TLS in an end-to-end manner, and some handheld devices have adopted Web browsers that support SSL/TLS. In comparison, MC-SSL tries to solve the problems by extending SSL/TLS, which is the de facto security protocol in the Internet. Second, ITLS requires a handheld device to do encryption/decryption twice as much as SSL or WTLS does. That would further increase the processor time and corresponding delays. Third, since the WAP gateway in ITLS cannot get any plaintext message by means of decryption, it cannot perform functions at application layer such as content transformation.

4.2 Independent SSL Connections

A simple approach to improve the end-to-end security of SSL chain model is to simultaneously have an end-to-end SSL connection and an SSL proxy chain between **C** and **S.** Figure 4-1 illustrates this idea. Both connections are independent from each other. Sensitive data are supposed to go through only the end-to-end connection. This is an intuitive solution adopted by some applications. For example, Kennedy [16] proposed an interesting proxy architecture that allows application-specific proxies (mobile codes) to be deployed automatically at the request of a client. That proxy architecture adopts an independent SSL chain as a secure communication channel between **C** and **S**.

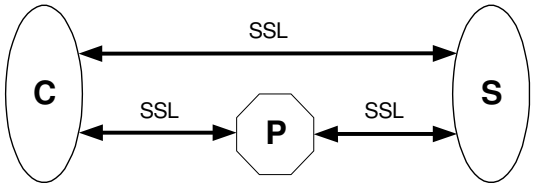

Figure 4-1: Independent SSL Connections

This approach certainly improves the end-to-end security when an application proxy is required. It is possibly secure enough for some applications if all sensitive data are transported using the end-to-end connection. However, it is generally insecure because of the authentication problem, that is, **P** is authenticated to **S** as **C** by using **C**'s authentication data such as user id and password. In general, **P** cannot get authorization from **S** using its own identity data even though that is a PKI certificate. In contrast, proxies in MC-SSL are negotiated through the end-to-end channel before **C** starts to set up a proxy channel with **S**. Moreover, **P** can then use the session id that is received from **C** as effective authentication data. In brief, a proxy in MC-SSL is authenticated as a proxy, not as a client.

In Figure 4-1, if **P** is not a highly trustworthy proxy, it can impersonate **C** to do sensitive transactions that should be performed only through the end-to-end connection. Besides, most servers today still authenticate their clients by user id and password. If a proxy keeps a client's id and password, it can impersonate a client in unconstrained fashion. There are some solutions for **C** to avoid exposing id and password, such as sharing symmetric session keys between **C** and **S** to a proxy. However, no matter how the authentication problem can be improved, a proxy can still impersonate **C** in a particular session if the SSL chain is independent of the end-to-end connection.

4.3 Changing Cipher Suite in SSL

SSL/TLS has defined many different cipher suites. The following table lists three cipher suites defined in RFC 2246 [1]:

CipherSuite	Is Exportable	Key Exchange	Cipher	Hash
TLS_NULL_WITH_NULL_NULL	Y	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	Y	RSA	NULL	MD5
TLS_RSA_WITH_3DES_EDE_CBC_SHA	N	RSA	3DES_EDE_CBC	SHA

These three cipher suites are very different in terms of security protection. TLS_NULL_WITH_NULL_NULL is a null cipher suite that does not provide any protection. TLS_RSA_WITH_NULL_MD5 is a cipher suite that uses RSA for authentication and key exchange and MD5 for MAC, but no cipher for application data encryption. TLS_RSA_WITH_3DES_EDE_CBC_SHA stands for a combination of RSA for key exchange, 3DES_EDE_CBC [2, 8] for block encryption, and SHA for MAC.

The working cipher suite for a particular SSL session is negotiated by the SSL handshake protocol, which is briefly explained in Appendix A. A client initiates the handshake procedure with a ClientHello message, which carries a list of cipher suites that is acceptable for the client. A server responds with a ServerHello message, which specifies a cipher suite chosen by the server. As mentioned in Section 2.2, SSL allows changing cipher suite by doing a full handshake, which is shown in Figure A-2 (a). In SSL 3.0, both client and server can initiate a full handshake, but only client can do so in TLS 1.0. If a server cannot decide when to change cipher suite, this approach of changing cipher suite by renegotiation is not

practically useful because most information is located at the server side in the Internet. Moreover, this approach is impractical even for SSL 3.0. There are several strong reasons to against this approach. First, SSL client and SSL server do not have enough information such as security policies and device capabilities to decide whether a new cipher suite is appropriate. Second, a full handshake including authentication and key exchange is very inefficient for changing cipher suite. Third, messages in opposite directions often need different level of protection, but it is very inefficient to change cipher suite for each request or response by doing a full handshake. In contrast, MC-SSL does not have these drawbacks.

4.4 Data Compression

Besides reducing storage space and bandwidth consumption, data compression has a good side effect, that is, reducing cryptographic operations because traffic volume is decreased as a whole. Data compression can be carried out at application or transport layer. Both the handshake and record protocol of SSL have mechanisms to support data compression before encryption and hash are applied to data. However, the specifications of SSL do not officially define any algorithm of data compression, and few implementations of SSL have built-in algorithms of data compression probably because there is no guarantee of interoperability. Consequently, SSL does not support data compression in reality. This is probably because it is generally better to perform data compression at application layer than at transport layer. In fact, it is the best for different types of data to adopt different algorithms of data compression. For example, JPEG (Joint Photographic Experts Group) format is for compressing static pictures, MP3 (MPEG Layer 3) format for music, and ZIP or GZIP (GNU ZIP) for lossless compression of ordinary data. Currently, most Web servers still do not

provide HTTP or HTML compression by using algorithms such as GZIP although popular web browsers such as IE and Netscape is able to support “Content-Encoding” and GZIP. By using GZIP to perform HTTP compression, webmasters can see a 150-160% increase in Web server performance, and a 70% - 80% reduction in HTML/XML/JavaScript bandwidth utilized [17]. Certainly, such a method can reduce HTTPS traffic as well, which in turn reduces cryptographic operations at both client and server side.

4.5 An SSL Extension for a Cleartext Channel

Portmann and Seneviratne proposed a simple extension to TLS in order to obtain an extra cleartext channel [4]. As shown in Figure A-1, TLS has four record types for Handshake Protocol, CCS (Change Cipher Spec) Protocol, Alert Protocol, and Application Data Protocol, respectively. The proposed extension defines a new record type called Cleartext Application Data (CAD). The data payload carried by CAD is in the form of cleartext, which means no encryption or MAC. The purpose of CAD is for transporting non-sensitive data, such as HTML tags, so that application proxies can adapt them to the needs of mobile devices; meanwhile, sensitive data can be transported using the original secure channel of TLS.

This new record type adds a cleartext channel to a TLS connection. To some degree, this channel is equivalent to a cleartext end-to-end channel in MC-SSL; however, this CAD channel is permanent as well as independent, which makes it insecure even if no sensitive data goes through it because a person-in-the-middle can inject data into the channel without detection. Without MAC or digital signature, a cleartext channel is inherently unable to prevent information tampering or injection, and non-sensitive data could be displayed side by

side with sensitive data. For this reason, an obvious drawback of CAD is that it is always there even if it is considered both unnecessary and insecure in some application cases. Moreover, it is not capable of creating channels other than a cleartext channel. In comparison, MC-SSL can provide a variety of channels including proxy channels and end-to-end channels that are equipped with various cipher suites; moreover, every channel is securely negotiated among client, server, and proxies. For example, in case that a cleartext channel is insecure, client and server should not negotiate a cleartext channel; instead, they can set up a MAC channel with or without proxies. A MAC channel can provide data integrity by using SHA-1 or MD5 hash algorithms.

4.6 Acceleration of Cryptographic Operations

Efficient cryptographic algorithms can achieve equivalent security protection with less computation. For example, AES (Advanced Encryption Standard) [2, 8], also called Rijndael, is an efficient algorithm for symmetric encryption. It is roughly twice as fast as 3DES [3]. For asymmetric cryptography, ECC (Elliptic Curve Cryptography) [2, 8] is much faster than RSA when generating a digital signature or decrypting data using a private key [18]; however, RSA is mush faster than ECC when verifying a signature or encrypting data using a public key. For this reason, handheld devices prefer RSA to ECC because they perform more public key operations than private key operations. Besides developing efficient algorithms, algorithms can be optimized to minimize their computational requirements [19, 20]. Although the advancement of cryptographic algorithms is very important, the speed of advancement and application is very slow. For example, it took three years from 1997 to 2000 for NIST (National Institute of Standards and Technology) in USA to choose AES as

the symmetric encryption standard assumed for next 20-30 years. Moreover, even though 128-bit keys are considered sufficiently secure at present, the development of cryptographic analysis and computer technologies will require the increase of key length.

Another approach is by using a co-processor specifically designed for cryptographic operations [21, 22]. Some companies even developed specific SSL chips and cards for SSL-enabled servers. For example, an SSL off-load NIC (www.layern.com) is capable of establishing 10,000 new SSL handshakes per second and up to 600 Mbps data encryption. These hardware approaches are very effective, but they certainly increase the cost of a device, and likely consume more battery power for handheld devices.

4.7 XML-based Solutions

This section first briefly introduces XML-based security solutions including XML Security [13, 14] and Web Services Security (WSS) [23, 24], and then compares them with MC-SSL.

XML Security is a set of core specifications developed by W3C (www.w3c.org). XML Security includes XML Signature [13] and XML Encryption [14], which defines XML syntaxes to represent cryptographic primitives such as encryption, hash, digital signature, and digital certificate. Based on XML Security, OASIS (Organization for the Advancement of Structured Information Standards) is developing WSS (Web Services Security), SAML (Security Assertion Markup Language), XrML (Extensible Rights Markup Language) [25].

WSS is a security framework that unites a number of existing and emerging specifications for securing Web services. By leveraging XML and the Web services model, this framework aims to be extensible, flexible, and interoperable. Figure 4-2 illustrates the roadmap of WSS [23], which shows that the development of WSS is an ongoing process. WSS is currently at

the stage of WS-Security, which mainly defines the enhancement to SOAP (Simple Object Access Protocol) messaging to provide message integrity and message confidentiality [24]. WS-Security also defines how to encode security tokens such as X.509 certificates, Kerberos tickets, username/password tokens, SAML tokens, and opaque encrypted keys [26, 27, 28]. Message integrity and message confidentiality are provided by leveraging security tokens with XML Signature and XML Encryption.

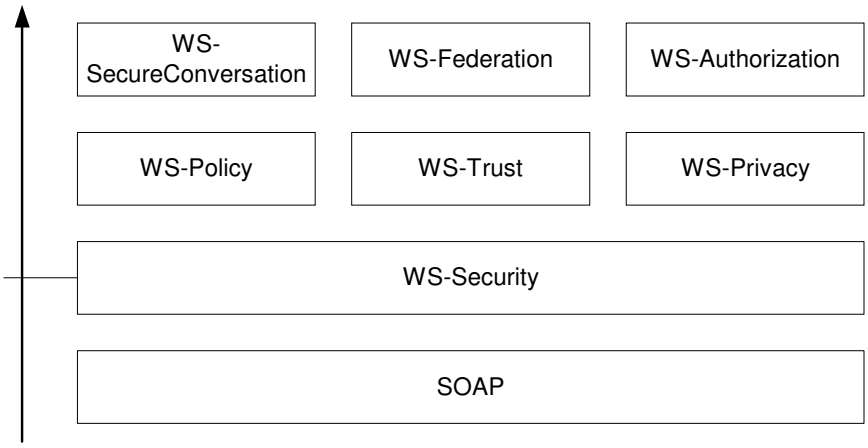

Figure 4-2: The Roadmap of Web Services Security [23]

MC-SSL is a complete and compact security protocol under application layer, and it is able to provide authentication, key exchange, and data security for client-server applications with or without proxies. XML-based solutions are different from MC-SSL in several aspects. First, they are not self-contained security protocols for client-server applications; otherwise, WSS cannot have extensibility and flexibility to integrate with existing or new security technologies. Most importantly, they are not able to perform authentication and key exchange among client, server, and proxies by themselves. Second, they do not have mechanisms to negotiate different types of channels described in Section 3.1, while MC-SSL is designed to

do so. Third, XML-based solutions generally belong to application layer, and they require client and server to support XML, XML Security, SOAP, and so on. MC-SSL is located under application layer, and it works for a variety of applications including Web services.

Besides the above differences, MC-SSL has some advantages. First, MC-SSL is more efficient than XML-based solutions: XML-based solutions often transform binary data into text data using base64 encoding, and add many tags. That could significantly increase network traffic. Second, MC-SSL is extended from SSL, and SSL is the *de facto* security protocol. Therefore, the change from SSL to MC-SSL is much smaller. On the other hand, XML-based solutions have their advantages. They are flexible and extensible because of XML, and they are able to combine existing and emerging security techniques to meet requirements of a specific application.

MC-SSL is not proposed to compete against any XML solution. In reverse, XML solutions can be combined with MC-SSL to support end-to-end security and selective security. For example, an application can make use of MC-SSL to do authentication, key exchange, and channel negotiation among client, server, and proxies, and take advantage of the flexibility of XML Security to represent complex encryption, hash, and signature for some application data.

4.8 Security Analysis and Development of SSL

Since MC-SSL is extended from SSL, the development and improvement of SSL is a good reference for designing and implementing MC-SSL. Netscape Communications Corp. published SSL 1.0 in 1994, and then SSL 2.0 and SSL 3.0 in 1995 and 1996, respectively. In 1999, IETF adopted SSL 3.0 with some minor modifications, and changed its name to TLS

(Transport Layer Security) 1.0 [1]. In 2003, RFC 3546 [29] introduced some extensions to TLS 1.0. WAP Forum made some minor changes to TLS for WAP devices, and created WTLS (Wireless Transport Layer Security). From SSL 1.0 to TLS 1.0, SSL was improved mainly by eliminating security vulnerabilities. After that, researchers started extending SSL to meet new requirements emerging from various clients, servers, and networks.

SSL has undergone nearly ten years of scrutiny by the security community. Security algorithms and protocols such as SSL and MC-SSL need rigorous security analysis to eliminate hidden vulnerabilities. Security analyses that improved security protocols such as SSL and IPSec are used as references for the protocol design of MC-SSL.

Bellovin [30, 31] found some attacks on IPSec protocol including cut-paste attack, chosen plaintext attack, and probable plaintext attack, which inspired other researchers to do similar analysis on SSL and WTLS. Wagner and Schneier [32] gave a detailed analysis of the cryptographic strength of the SSL 3.0 protocol. They found a number of minor flaws and several new attacks such as traffic analysis of request lengths, change-cipher-spec-dropping, and key-exchange-algorithm-spoofing; however, these flaws are not very serious, and they can be easily corrected without overhauling SSL. Mitchell et al [33] analyzed SSL 3.0 using a finite-state verification tool named Murφ. They analyzed a sequence of protocols that incrementally approximate to SSL 3.0, which demonstrates that the iterative process of design and analysis is effective to improve a security protocol. Paulson [34] analyzed TLS 1.0 using a theorem prover named Isabelle. Bleichenbacher [35, 1] found a chosen ciphertext attack against security protocols using PKCS #1 v1.5, which is the de factor implementation of the RSA algorithm. Saarinen [36] found a number of security problems of WTLS

including a chosen plaintext attack, a datagram truncation attack, a message forgery attack, and a key-search shortcut for some exportable keys.

Chapter 5

Protocol Design

This chapter presents MC-SSL protocol that implements the multiple-channel model described in Chapter 3. MC-SSL protocol consists of Initial Handshake Protocol, Primary Proxy Channel Protocol, Secondary Channel Protocol, Channel Cancellation Protocol, Alert Protocol, Abbreviated Handshake Protocol, and Application Data Protocol. Initial Handshake Protocol can establish the primary end-to-end channel, which is the first channel in an MC-SSL session, and convey information about security policy and device capabilities. Primary Proxy Channel Protocol is responsible for negotiating a primary proxy channel. Secondary Channel Protocol is able to set up secondary channels. Channel Cancellation Protocol cancels any existing channel during a session. Alert Protocol conveys errors and warnings. Abbreviated Handshake Protocol can resume a cached MC-SSL session. Application Data Protocol transports data using an end-to-end channel or a proxy channel. In addition, this chapter describes MC-SSL protocol architecture in the beginning, and discusses the security and usage of MC-SSL in the end.

5.1 Protocol Architecture

In Figure 5-1, the left part shows the Internet protocol stack with SSL, and the right part shows the protocol stack with MC-SSL. MC-SSL protocol is deliberately designed to consist of two layers: the lower layer is SSL, and the upper layer is a new layer inserted between SSL and the application layer, which provides the application layer with new functions that

SSL lacks. The upper layer is responsible for channel control of multiple channels, and thus named as “MC” (Multiple Channel) layer. MC-SSL protocol has two types of messages: channel control message and data message. The former is for negotiating and controlling channels, and the latter is for transporting data for the application layer.

MC-SSL protocol is designed to reuse SSL protocol as much as possible as long as it can implement the model of MC-SSL. SSL is therefore the basis of MC-SSL in terms of protocol design, software implementation, and security strength. The lower layer of MC-SSL, i.e. SSL, is kept untouched unless secondary channels, i.e. multiple cipher suites, are implemented. To support secondary channels, a new field of channel id is added into SSL messages. An alternative is to design a handshake protocol in the upper layer to change cipher suites so that one do not have to change SSL; however, it is not as efficient as adding channel id in some cases. Section 5.6 presents secondary channel protocol.

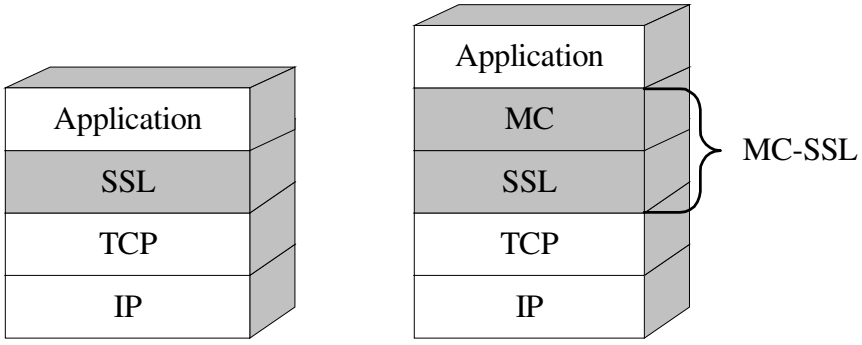

Figure 5-1: Two-layer Architecture of MC-SSL

5.2 Initial Handshake Protocol

Initial Handshake Protocol can set up the first channel, namely the primary end-to-end channel, in an MC-SSL session. The handshake process is illustrated in Figure 5-2. First, an

SSL session is established between **C** and **S**. After that, **C** and **S** exchange a pair of hello messages to initiate an MC-SSL session. Both hello messages, CLIENT_HELLO and SERVER_HELLO, carry the following information: protocol version, session id, MAC key and the hash algorithm for end-to-end MAC. Protocol version is the MC-SSL version number of **C** or **S**. Session id is a cryptographically random string that is generated by server to identify an MC-SSL session. MAC key is used by Application Data Protocol to generate end-to-end MAC. Please refer to Appendix B.3 for the message formats of Initial Handshake Protocol.

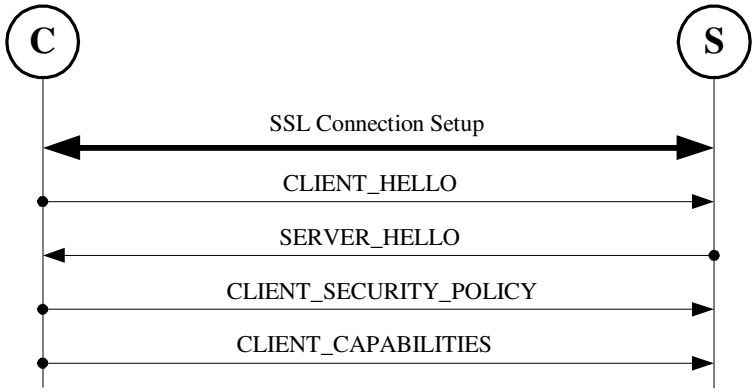

Figure 5-2: Initial Handshake Protocol

C then sends its security policy and device capabilities to **S** by CLIENT_SECURITY_POLICY and CLIENT_CAPABILITIES message. Security policy may define whether a proxy is allowed when **C** or **S** delivers information that has a certain level of sensitivity. Device capabilities include hardware and software information of a device such as CPU, power, memory, screen resolution, OS, browser capabilities, etc. With such information about **C** and itself, **S** is expected to make right suggestions about what proxy channels and secondary channels are needed.

5.3 Proxy Channel Protocol

A channel can be categorized by different criteria. According to whether a channel has application proxies or not, it can be a proxy channel or an end-to-end channel. Whether or not to be the first channel in a connection determines a channel to be a primary channel or a secondary channel. A primary channel is negotiated using SSL, and it is the backbone channel to negotiate and control secondary channels in the same connection. In the sample MC-SSL session shown in Figure 3-3, channel 1 is the primary end-to-end channel, and channel 4 is a primary proxy channel; channel 2 and 3 are secondary end-to-end channels, and channel 5 is a secondary proxy channel.

The Initial Handshake Protocol can set up the primary end-to-end channel. The Proxy Channel Protocol described in this section is able to negotiate primary proxy channels, the backbone channels to negotiate secondary proxy channels. Let us first delve into the protocol for single-hop proxy channels, and then extend the protocol to the general case, that is, multi-hop proxy channels.

5.3.1 Single-Hop Proxy Channel Protocol

A complete protocol for establishing a single-hop proxy channel is illustrated in Figure 5-3. It includes four stages: **C**-**S** handshake, **C**-**P** handshake, **P**-**S** handshake, and feedback of negotiation. Apart from the SSL channel between **C** and **S**, which is set up by the Initial Handshake Protocol, a single-hop proxy channel needs to set up two more SSL channels.

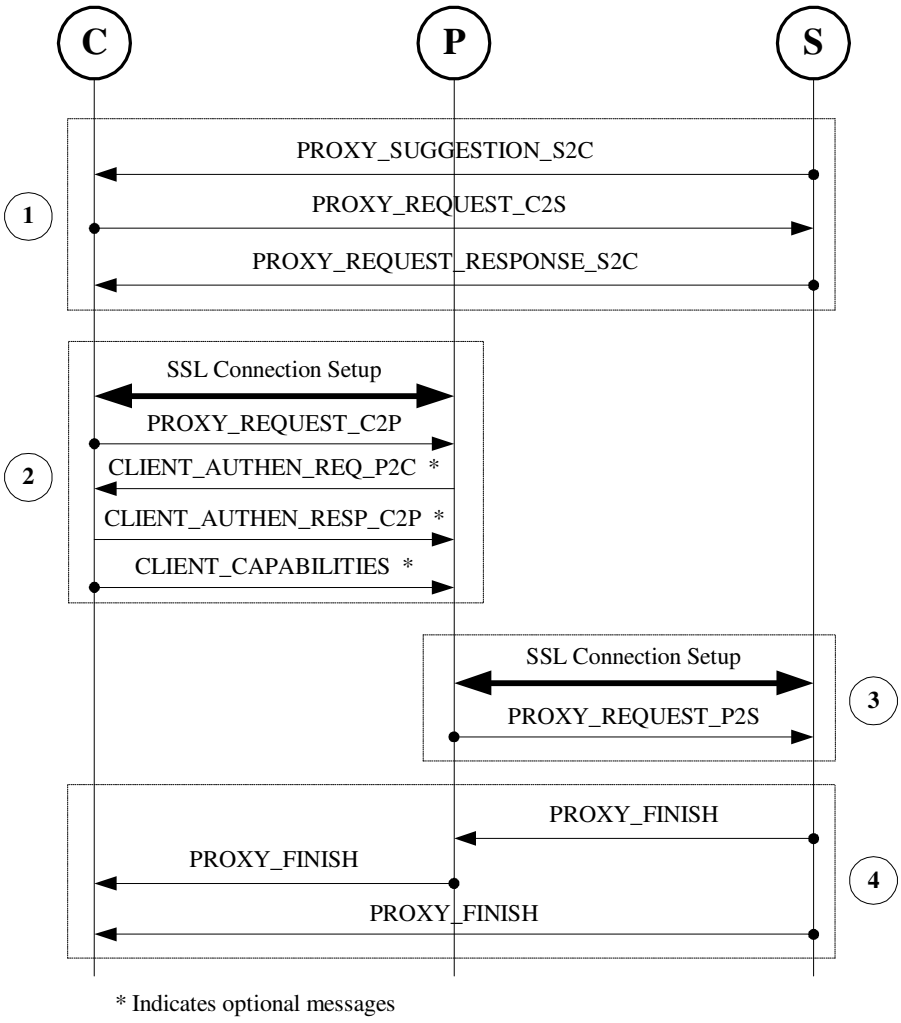

Figure 5-3: Primary Proxy Channel Protocol

S or **C** can start the negotiation of a proxy channel any time after the initial handshake of MC-SSL. In part 1 of Figure 5-3, **S** starts the negotiation by sending a proxy suggestion message (PROXY_SUGGESTION_S2C) to **C**, which contains information about the proxy channel, such as the purpose of the proxy, the DNS name and the certificate of the proxy, and channel direction. **C** then sends a proxy request message (PROXY_REQUEST_C2S) back to **S**, which likewise carries information about the proxy channel. In this message, **C** can use the proxy and channel parameters suggested by **S**, change some parameters, or even use a

different proxy. **S** responds with the proxy request response message (PROXY_REQUEST_RESPONSE_S2C) to give its final decision. Please refer to Appendix B.4 for the message formats of Primary Proxy Channel Protocol.

Both **C** and **S** can initiate a **C**-**S** handshake to negotiate a proxy channel. The **C**-**S** handshake can be interrupted if **C** or **S** decides that a proxy is not secure or necessary. In this handshake process, both **C** and **S** have a right to suggest, change, or veto channel parameters including the proxy and the traffic direction. Further, proxies recommended by **C** and **S** for different purposes can be combined to form a multi-hop proxy channel. It is up to **C** and **S** to implement their own decision process for a particular application.

The **C**-**P** handshake starts with the **C**-**P** SSL handshake. After that, **C** sends a proxy request message (PROXY_REQUEST_C2P) to **P**, which carries the following information: the session id, proxy services needed, channel direction, authentication methods preferred by **C**, handshake type, the IP address and port number of **S**. CLIENT_AUTHEN_REQ_P2C and CLIENT_AUTHEN_RESP_C2P are a pair of messages for **P** to authenticate **C**. The former tells **C** the required authentication method, such as user id/password, challenge/answer, or PKI certificate, and the latter returns the corresponding authentication data to **P**. If **P** does not require authentication or it can authenticate C in a special way, these two messages may be omitted. In addition, if **P** needs to know **C**'s capabilities to perform its service, a CLIENT_CAPABILITIES message will follow. In Figure 5-3, these optional messages are indicated by an asterisk beside their names.

If **C** passes the authentication, **P** will set up an SSL connection with **S,** and then send a proxy request message (PROXY_REQUEST_P2S) to **S**. This message carries a session id for

S to bind the proxy channel with the end-to-end channel in the same session. The last three messages in Figure 5-3 return the result of negotiation.

In the **P**-**S** handshake stage, there is no message for authentication. If **P** is public or commercial proxy server, **P** can authenticate itself using its PKI certificate in the handshake process of SSL. However, as discussed in Section 3.2.2, **P** could be a computer at home, which usually does not have a PKI certificate. How can **P** authenticate itself in that kind of situation? In fact, session id alone is good enough to be **P**'s credential. Because session id is a cryptographically random string or number generated by **S**, and it is exchanged as a secret using SSL (primary channels) among **S**, **C**, and **P**, it becomes a good security token for **P**. Therefore, **P** does not have to possess a certificate acceptable to **S**. This is a very useful feature because a handheld device can designate a computer at home as a proxy. For a handheld device to authenticate a home computer in the SSL handshake of the **C**-**P** handshake stage, a user can simply generate a certificate and its accompanying private key on the home computer, and import this "homemade" certificate into his handheld device before using the home computer as a proxy.

5.3.2 Extensions to Support Multi-hop Proxy Channels

A single-hop proxy channel is normally simpler and securer than a multi-hop one. As discussed in Chapter 3.2, a proxy "cluster", which chooses one proxy as the representative to interact with **C** and **S**, can reduce the number of proxies in a proxy channel. However, multi-hop proxy channels are sometimes unavoidable due to administrative or security reasons. For instance, **S** assigns a proxy that is responsible for the content transforming of **S**'s Web pages,

and meanwhile, **C** requires Web pages to go through an application firewall for virus scanning and filtering. In that case, there are two proxies in the proxy channel.

First, let us consider the simplest way to extend the protocol described in the previous section: we can iteratively reuse the **P**-**S** handshake, i.e. stage 3 in Figure 5-3, on any two neighbouring proxies in Figure 3-2, for instance, from $\mathbf{P}_i$ to $\mathbf{P}_{i+1}$. Similar to a single-hop proxy channel, this forward process starts at **C** and ends at **S**. The proxy request messages need small changes: they have to carry parameters of all proxies, not just one. In the **C**-**S** handshake, **C** and **S** need to exchange the information about DNS names, listening TCP ports, and even certificates (or their URLs) of all proxies. Likewise, the proxy request message in the **C**-$\mathbf{P}_1$ handshake is extended to contain information of multiple proxies. After the **C**-$\mathbf{P}_1$ handshake is done, $\mathbf{P}_1$ connects to $\mathbf{P}_2$ as the **P**-**S** handshake in the single-hop proxy channel protocol does. This forward process continues until the last proxy $\mathbf{P}_n$ interacts with **S**. The feedback process, i.e. stage 4 in Figure 5-3, can be easily extended for a multi-hop proxy channel without much change.

After this extended handshake process is completed, every entity in Figure 3-2 has authenticated its two neighbours if we topologically look at the structure in Figure 3-2 as a ring. As a result, **C** can trust proxies from $\mathbf{P}_2$ to $\mathbf{P}_n$, and **S** can trust proxies from $\mathbf{P}_1$ to $\mathbf{P}_{n-1}$, both in a transitional way. In the case of a single-hop proxy channel, there is no need for transitional trust because **C**, **P**, and **S** have directly authenticated one another. Although simplest protocol extension requires transitional trust, it should be good enough for many applications because proxy channels in MC-SSL are supposed to transport relatively non-sensitive data. Sensitive data should be transported using end-to-end channels.

We can further enhance this transitional trust model by appending new message interactions to the above negotiation process. The enhancement is proposed for the following needs. First, proxies other than $\mathbf{P}_1$ may also require authenticating **C** by themselves for accounting or other reasons. Second, **S** or **C** may want to directly authenticate all proxies.

If **C** and all proxies have their own certificates, we can have a protocol that consists of two stages illustrated in Figure 5-4. The rationale is to generate a random string or number, and ask **C** and all proxies to "sign" the random string using their private keys. The "signatures" (i.e. verification data) are circulated and used for identity verification.

Figure 5-4 (a) shows the first stage, which tries to generate a random string that is a concatenation of random strings produced by all the entities in the circle. The resultant string can be denoted as follows:

$$\boldsymbol{R} = \boldsymbol{R}_C + \boldsymbol{R}_{P1} + \boldsymbol{R}_{P2} + \ldots + \boldsymbol{R}_S,$$

$\boldsymbol{R}_X$ denotes a 32-byte *cryptographically random* string generated by entity $\boldsymbol{X}$, and '+' denotes the concatenation of two strings. This process starts and ends at **C**. The message from **C** to **P1** contains only $\boldsymbol{R}_C$, but at last **C** receives the complete random string from **S**. By collectively generating a random string, no entity can be "deceived" by other entities to accept a random string that is not truly random because at least 32 bytes of $\boldsymbol{R}$ is generated by itself. In other words, each entity creates a random challenge $\boldsymbol{R}_X$. For efficiency, all challenges are merged into a single challenge $\boldsymbol{R}$, which is then signed by entities using digital certificates in the second stage. This method is actually an extension of SSL, in which only two entities (i.e. **C** and **S**) are involved in the ring.

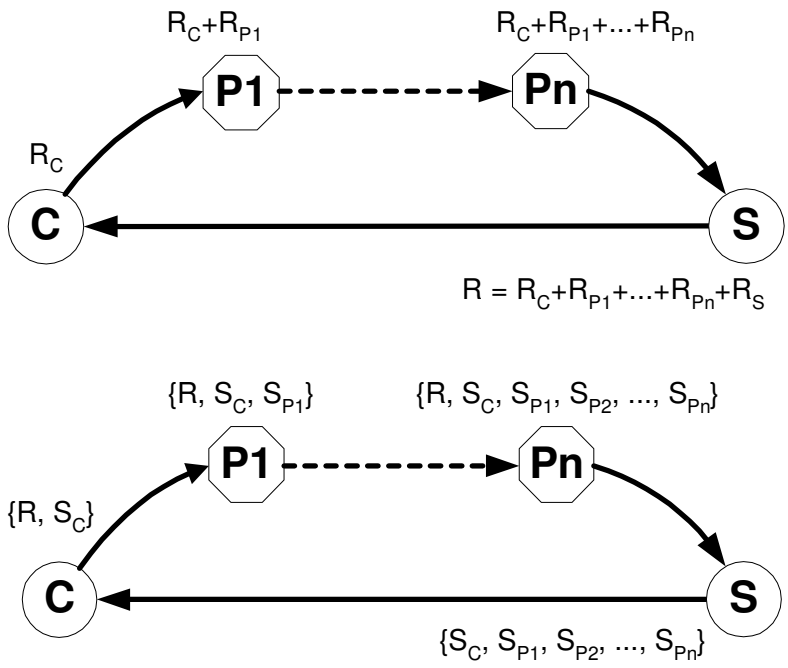

Figure 5-4: The enhanced authentication: (a) first stage, (b) second stage

Figure 5-4 (b) shows the second stage. $\boldsymbol{S}_X$ denotes the signature signed by ***X***. **C** sends **P1** a message that contains the random string (**R**) generated in the first stage, the certificate (or its URL) of **C**, and the digital signature ($\boldsymbol{S}_C$) generated upon ***R*** with **C**'s private key. The signature can prove that **C** is the owner of the certificate. Each proxy adds a new signature using its private key; meanwhile, each proxy can verify **C**'s identity using **C**'s certificate. When the message arrives at **S**, it has collected the signatures of all proxies, and therefore **S** can verify all of them using their certificates. **S** can also forward them to **C** if **C** wants to verify them as well. Section 5.3.1 claims that the proxy in a single-hop proxy channel may not need a PKI certificate since session id is randomly generated by **S**, but that is not true for a multi-hop proxy channel because any two neighbouring proxies have to authenticate each other. As discussed in Section 3.2.3, a multi-hop proxy channel is more complex than a single-hop one, and thus need more support from security infrastructure such as PKI.

For the first stage, we can add a new field in proxy request messages and the **S**-**C** proxy finish message to carry the random string, a flag field to indicate if **S** or **C** requires verification of proxies' certificates, and another flag field to indicate if any proxy requires verification of **C**'s certificate. If no verification is requested, the second stage will not start. For the second stage, we need a new message called CP_VERIFICAION to carry forward all the necessary information, and this protocol ends with a verification finish message (CP_VERI_FINISH) from **C** to **S**.

The protocol described above is about the authentication of proxies and the client. We may also need to consider the security issues of transporting and processing application data through multiple proxies. For example, an application might require that every chunk of data go through every designated proxy. To achieve this kind of data authenticity, we can ask proxies to "sign" a data chunk using their private key or MAC key. However, that introduces heavy computational burden because **C** or **S** has to deal with every chunk of data. The cost seems too high for the reason we mentioned before: a proxy channel is not supposed to transport highly sensitive data unless all the proxies in the channel are highly trusted.

5.4 Application Data Protocol

The purpose of Application Data Protocol is to transport application data between **C** and **S**. There are two ways to do this: one is through the end-to-end channel as shown in the A part of Figure 5-5; the other is through the proxy channel under the control of the end-to-end channel, as shown in the B part of Figure 5-5. Figure 5-5 only shows that **S** sends data to **C**. The transmission in the opposite direction uses the same messages.

To use the end-to-end channel, the sender encapsulates data into APP_DATA_DIRECT message and sends to the receiver through the end-to-end channel. To transport a piece of content through the proxy channel, three messages are involved: APP_DATA_TO_PROXY, APP_DATA_FROM_PROXY, and APP_DATA_CONTROL_PROXY. The sender wraps the content into APP_DATA_TO_PROXY, and sends it to **P**. After processing the content, **P** generates APP_DATA_FROM_PROXY, and forwards the result to the receiver. Besides, the sender sends an APP_DATA_CONTROL_PROXY message to the receiver in order to control the behaviour of **P**.

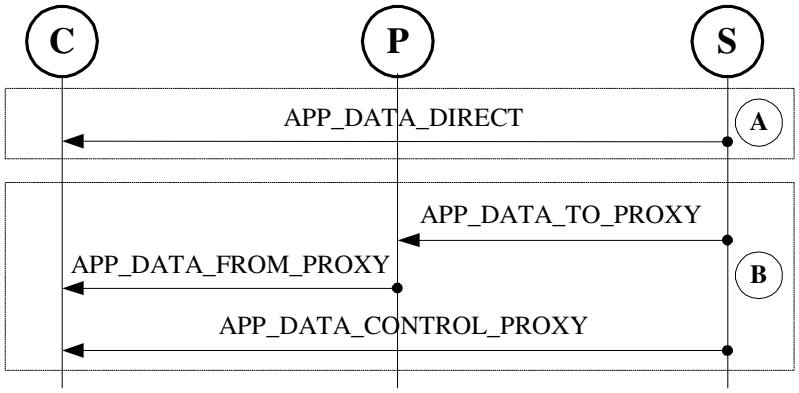

Figure 5-5: Application Data Protocol

APP_DATA_TO_PROXY message contains such fields as content, processing request, and change restriction. Field content can be a complete or fragmental piece of content. Processing request tells **P** how to process the content. Change restriction indicates if the content is unchangeable, modifiable, or discardable. APP_DATA_FROM_PROXY includes such fields as content and result. The former is the processed content. The latter outlines the processing result. Please refer to Appendix B.5 for the message formats of Application Data Protocol.

The purpose of APP_DATA_CONTROL_PROXY message is to control the behaviour of **P**. It conveys information such as proxy channel id, content attributes, change restriction, and MAC. Proxy channel id indicates which proxy channel the corresponding APP_DATA_TO_PROXY message has gone through. Field content attribute is a semicolon-separated compound string that describes attributes of content such as content types. The benefit of knowing content types is that a receiver can detect whether or not **P** has injected new types of contents, which could be malicious code. For example, **P** should not be allowed to add JavaScript or any other potentially malicious code into a pure HTML page. Change restriction indicates if the content is unchangeable, modifiable, or discardable. MAC field is the MAC of the content carried in APP_DATA_TO_PROXY message, which is used by a receiver to verify the integrity of content if it is unchangeable. MAC is calculated with HMAC hash function described in RFC 2246 [1]. MAC keys and the hash algorithm for computing this MAC are negotiated by the hello messages in Initial Handshake Protocol.

When receiving data, **C** and **S** always wait for incoming data at the end-to-end connection. If an APP_DATA_DIRECT message arrives, they will get the data encapsulated in the message. If an APP_DATA_CONTROL_PROXY message arrives, they will switch to the proxy channel specified by the proxy control message, and get the data encapsulated in the APP_DATA_FROM_PROXY message. Therefore, a receiver will first read the proxy control message even though its corresponding data message arrives earlier at the connection of the proxy channel.

5.5 Alert Protocol

The purpose of Alert Protocol is to convey warnings and fatal errors in the process of protocol interactions, for instance, when an unknown or wrong message is received. Alert Protocol of MC-SSL is similar to SSL. Please refer to Appendix B.8 for the alert protocol. Currently, there are still many unsettled protocol details related to alerts. The Alert Protocol is expected to accurately define more alerts in the future. For example, in the Application Data Protocol, an application data message could never arrive at a client through a proxy channel. Sequence number of a message can help detect loss and repeat of a message. The client can also set up an alarm with a predefined timeout parameter after receiving a proxy control message. If loss or repeat of a message is detected, or a timeout event is triggered, corresponding alerts should be sent to a server and proxies.

5.6 Secondary Channel Protocol

The protocols we described to this point do not support multiple cipher suites (or secondary channels) in a point-to-point connection. The only channel we have in a connection is the primary channel provided by SSL. In this section, we discuss the protocol that negotiates and makes use of secondary channels. As defined in Section 3.1, every channel in MC-SSL can have its own cipher suite and channel direction.

5.6.1 Negotiation of Secondary Channels

The negotiation process of secondary channels is shown in Figure 5-5. Negotiating secondary end-to-end channels is shown in part **A**, and negotiating secondary proxy channels is shown in part **B**. This protocol adds two messages to MC-SSL: the secondary channel request message (SEC_CHAN_REQ) and the secondary channel response message

(SEC_CHAN_RESP). They are designed to carry requests and responses for multiple secondary channels in order to reduce message interactions if an application session needs multiple secondary channels. In addition, they are designed to travel through primary channels. In MC-SSL, channel control messages never travel through secondary channels so that channel negotiation or management is as secure as primary channels provided by SSL.

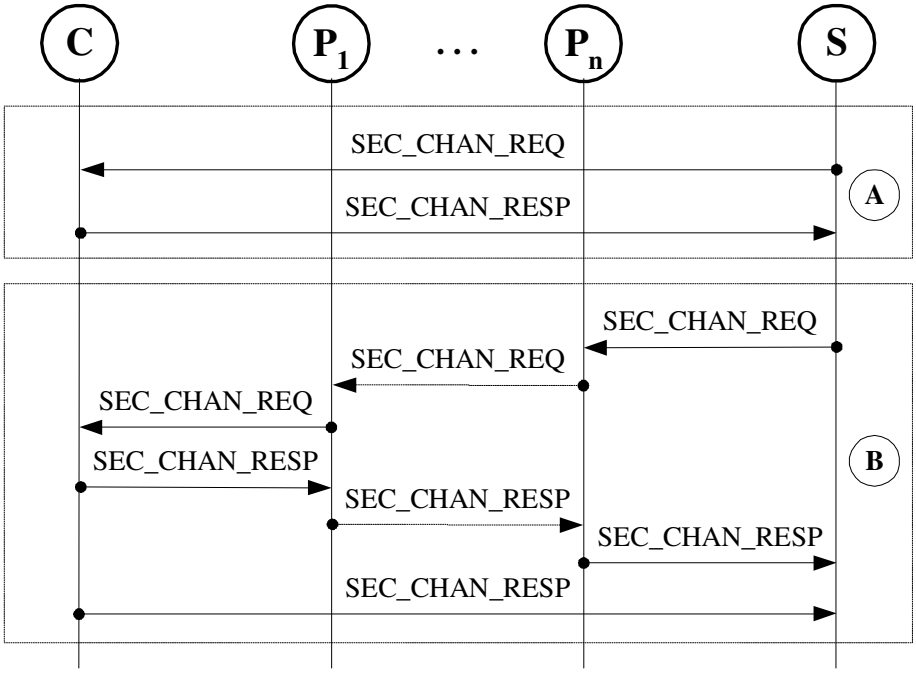

Figure 5-6: Negotiating Secondary Channels

An SEC_CHAN_REQ message can request multiple secondary channels. For each of them, it carries the following information: channel id, a list of cipher suites preferred by the message sender, and the channel direction. It also carries the channel id of the collaborative end-to-end channel if the secondary channel is a proxy channel. The collaborative end-to-end channel of a proxy channel is the channel that an APP_DATA_CONTROL_PROXY message travels through, and it could be the primary end-to-end channel or a secondary end-to-end channel. SEC_CHAN_RESP message carries the responses for channel requests in an

SEC_CHAN_REQ message. Please refer to Appendix B.6 for the message formats of Secondary Channel Protocol.

For MC-SSL to efficiently support multiple cipher suites, a small extension is introduced to SSL. This is discussed in the next section. However, it is possible that the actual SSL implementation at **C** or **S** does not support the extension. In that case, the above negotiation process will either fail or not start, and secondary channels will not be available.

5.6.2 Extending SSL to Support Secondary Channels

In order to support secondary channels, an easy way is to add a new field in the header of every SSL packet. The new field is channel id, the id of the channel through which a packet travels. When an SSL packet arrives, a receiver will use the cipher suite that corresponds to the channel id to decrypt and verify the payload that is encapsulated in the packet. Figure 5-7 illustrates an SSL packet, and the relations among channel id, a cipher suite, and payload. The mapping between channel id and a cipher suite is maintained by Secondary Channel Protocol in the previous section. In MC-SSL, the payload of an SSL packet contains an MC-SSL message. Whether the MAC in an SSL packet exists depends on whether the cipher suite has a hash algorithm. Channel id can realize multiplexing of multiple virtual channels inside an SSL connection.

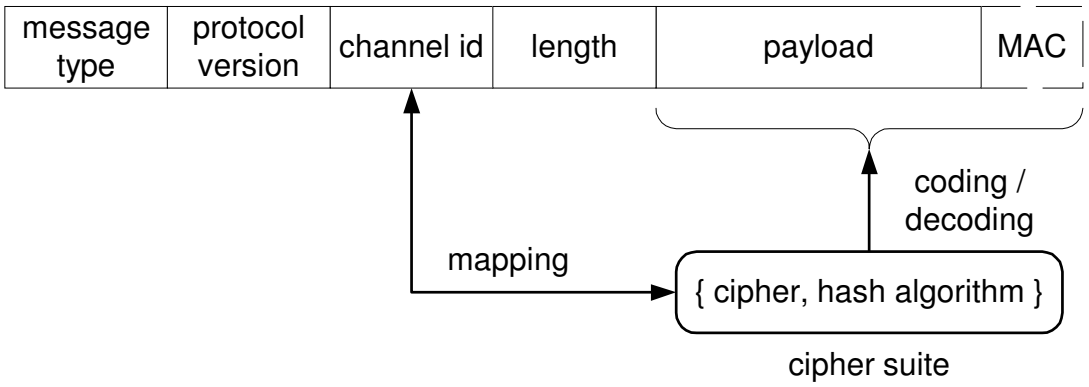

Figure 5-7: Adding Channel Id in an SSL Packet

Considering that other new extensions may emerge to require changing the packet format of the SSL record protocol as MC-SSL does, it is probably better to add a general extension field so that future extensions or options can be accommodated without changing SSL again. A similar extension mechanism has been introduced into the client and server hello messages of TLS handshake protocol. Please refer to RFC 3546 [29] for TLS extensions. In fact, many Internet protocols such as TCP and IP have fields in their packet formats for accommodating options or future extensions.

Because of the introduction of channel id, several changes regarding SSL protocol and its implementations follow. First, the calculation of MAC includes channel id if an SSL packet has a MAC field for the channel id not to be tampered without detection. Second, the implementation of SSL must be revised to choose the right cipher suite for each incoming packet according to the channel id field. Third, some API (Application Programming Interface) functions in SSL library are different than before: the write function has a channel id as an input parameter, and the read function returns as an output parameter the id of the channel from which the data comes. It is up to applications to decide how to use channels they set up.

5.6.3 An Alternative

Adding a channel id field is a simple approach to support multiple cipher suites, but the downside is that SSL protocol has to be slightly revised. This section gives an alternative for supporting secondary channels. This alternative can keep the protocol of SSL intact by "switching" the working cipher suites. The switching is realized by a simple handshake protocol at the upper layer of MC-SSL. Figure 5-8 shows an example that uses the simple handshake twice to change cipher suites for two opposite directions.

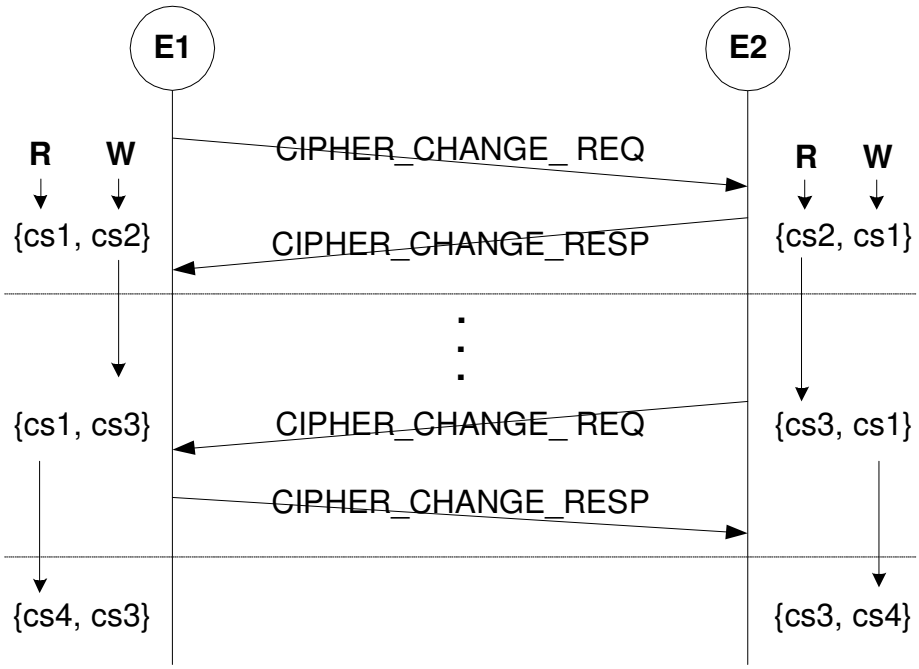

Figure 5-8: Handshake Protocol to Switch Cipher Suites

The handshake protocol consists of only two messages: a cipher change request (CIPHER_CHANGE_REQ) and a cipher change response (CIPHER_CHANGE_RESP). A request message carries the cipher suite that the sender will switch to. A response message confirms the request. Both these two messages have a MAC field to protect data integrity. The calculation of this MAC needs the MAC key and hash algorithms negotiated in the

Initial Handshake Protocol of MC-SSL. Note that this MAC field belongs to the upper layer of MC-SSL, and thus it is contained in the payload field in Figure 5-7.

In this approach, MC-SSL maintains two working cipher suites for the two opposite directions of an SSL connection. An SSL session has only one working cipher suite, which is used by both directions. In other words, SSL provides a duplex channel protected by a single working cipher suite. This approach changes a duplex channel into two simplex channels with two working cipher suites. Each endpoint maintains its own set of working cipher suites that can be denoted as {CS_R, CS_W}, where CS_R is for reading data and CS_W is for writing data. CS_W of one endpoint is CS_R of the other endpoint of the same connection. An endpoint is only responsible for changing its CS_W, namely the cipher suite for writing data. At the beginning of the process that is shown in Figure 5-8, **E1** has {cs1, cs2}, and **E2** has {cs2, cs1}. **E1** sends a request to change its CS_W, namely CS_R of **E2**. **E2** accepts the change. As a result, the working cipher suites of **E1** become {cs1, cs3}, and those of **E2** becomes {cs3, cs1}. Later on, **E2** switches its CS_W to cs4 using the same handshake protocol. Since two working cipher suites are maintained for two opposite directions of SSL, MC-SSL does not have to switch cipher suites in the case that requests use one cipher suite and responses use a different one. In brief, this method can reduce the frequency of switching cipher suites with the handshake protocol.

This handshake protocol at the upper layer of MC-SSL is much more efficient than the full handshake protocol of SSL/TLS if the latter is used to change cipher suite for an SSL session as described in Section 4.3. This approach has the advantage of keeping SSL protocol unchanged although it is not as simple and efficient as adding a channel id field; therefore, it is an alternative for realizing secondary channels in MC-SSL.

5.7 Restriction on Channel Directions

MC-SSL can restrict the data flow directions of channels. In Appendix B, four channel directions are defined: DUPLEX, C2S, S2C, and NONE. DUPLEX indicates a duplex channel. C2S or S2C indicates a simplex channel that only allows application data to flow from **C** to **S** or from **S** to **C**. NONE indicates an inactive channel, which inhibits any application data. Note that a primary channel can be used for channel control messages in both directions even if it is marked as C2S, S2C, or NONE. Restriction on channel directions only applies to application data messages.

As shown in Figure 5-9, the restriction on channel directions is enforced at the interface between the application layer and the MC layer of MC-SSL. If a channel is not duplex, it should reject receiving data from and/or delivering data to the application layer according to the specified channel direction. The underlying SSL does not know about channel directions of MC-SSL. In fact, there could be another duplex channel inside the same SSL connection, which allows both receiving and delivering application data.

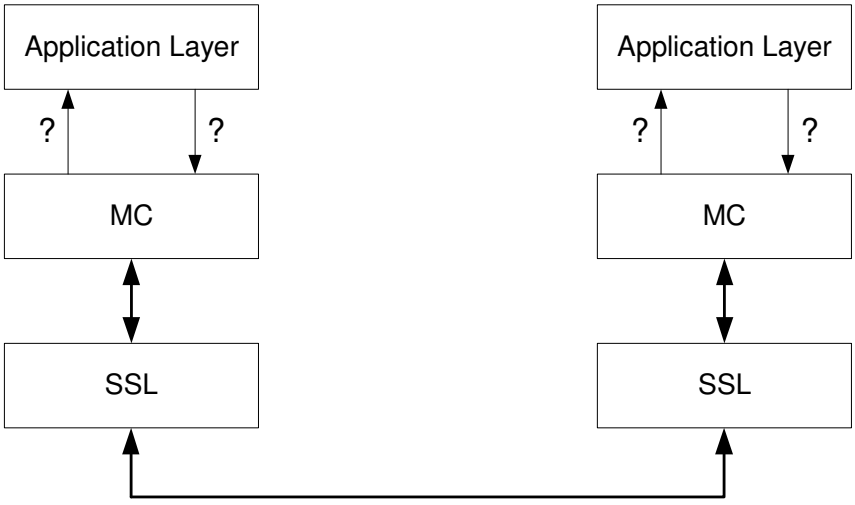

Figure 5-9: Implementation of Channel Directions

5.8 Channel Cancellation

Channel management of MC-SSL is flexible. **C** and **S** can negotiate a proxy channel or a secondary channel at any time. They can also cancel a channel at any time. The message interaction for channel cancellation is shown in Figure 5-10, which resembles Secondary Channel Protocol in Figure 5-6. Similar to Secondary Channel Protocol, the request message (CHAN_CANCEL_REQ) and the response message (CHAN_CANCEL_RESP) are able to cancel multiple channels in a single round.

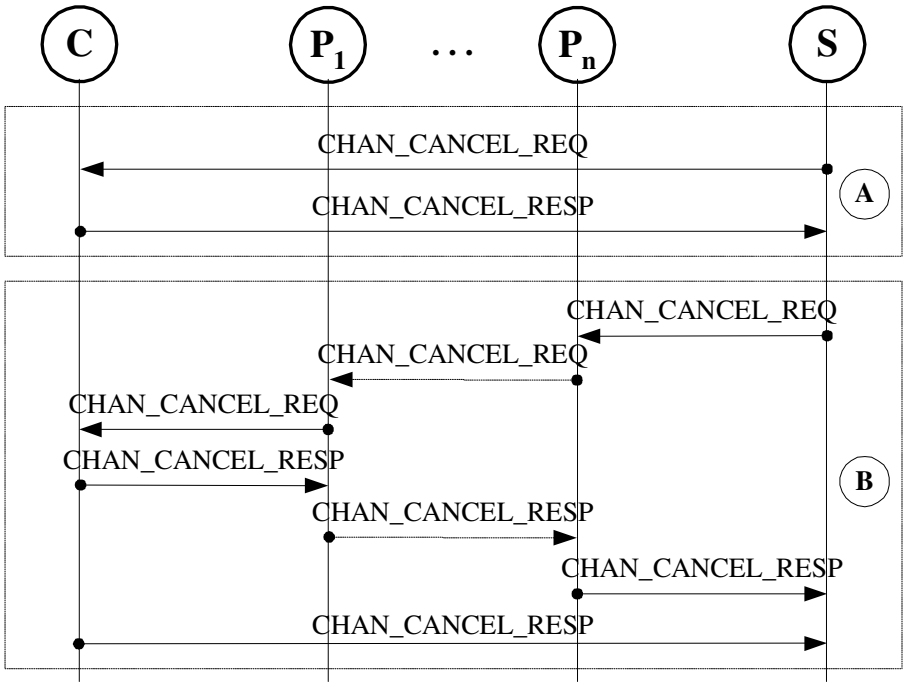

Figure 5-10: Channel Cancellation Protocol

5.9 Abbreviated Handshake

Like SSL does, MC-SSL supports the resumption of a cached session using an abbreviated handshake. Most messages in Initial Handshake Protocol (Figure 5-2), Proxy Channel Protocol (Figure 5-3), and Secondary Channel Protocol (Figure 5-6) are omitted. In Proxy

Channel Protocol, only the SSL handshakes, PROXY_SUGGESTION_S2C, PROXY_REQUEST_C2P, PROXY_REQUEST_P2S, and PROXY_FINISH messages are still required for an abbreviated handshake. In addition, when MC-SSL is doing an abbreviated handshake, the underlying SSL handshake is also abbreviated. Consequently, the key exchange, certificate exchange and verification, and most channel negotiation processes among client, server, and proxies are omitted. Therefore, an abbreviated handshake is much more efficient than a full handshake.

In the initial handshake of MC-SSL, after the abbreviated **C**-**S** SSL handshake, **C** and **S** only need to exchange a pair of client hello and server hello messages. The messages that convey the client's security policy and device capabilities are omitted. If the session id in the client hello message is not zero, it should be the session id of a cached MC-SSL session, which indicates an abbreviated MC-SSL handshake. If an abbreviated handshake succeeds, **C** and **S** will activate all end-to-end channels except those channels without MAC protection. It is risky to automatically activate a channel without data integrity protection.

Moreover, **C** and **S** can do an abbreviated handshake for a primary proxy channel. Similar to end-to-end channels, all secondary proxy channels subordinate to the primary proxy channel are activated together with the primary proxy channel, except those secondary channels without MAC protection.

Since secondary channels are activated together with their corresponding primary channels, they do not need specific abbreviated handshake process. After an abbreviated handshake is over, **C** and **S** can add or cancel channels.

5.10 Discussion

This section discusses the security and usage issues of MC-SSL and SSL after the whole MC-SSL protocol has been presented. Section 5.9.1 focuses on the security strength and concerns of MC-SSL proxy channels in comparison with SSL. Section 5.9.2 discusses typical usage of multiple cipher suites.

5.10.1 Proxy Channels

As pointed out in Section 2.1, when application proxies are needed, some applications allow **C** or even force **C** to use an SSL chain to access **S** through proxies such as a WAP gateway. Consequently, the end-to-end security completely depends on the degree of trustworthiness of proxies. In contrast, after having the secure primary end-to-end channel in place, MC-SSL lets **S** and **C** jointly decide whether or not to set up a proxy channel, what proxies best suit their needs, and how to securely use a proxy channel. Besides other advantages, this feature enables making such a decision after both parties have determined the degree of trust in each other, which could be a function of the corresponding credentials. For instance, if the certificate received from **S** is not a high-class certificate issued by a good Certificate Authority, **C** may assign an anti-virus or content scanning proxy as the last proxy in the proxy channel. It is a well-known problem that some Web pages automatically download malware onto computers.

The possibility of proxy misuse is another hidden issue of the SSL chain, which is not mentioned in Chapter 2. For instance, a user may mistakenly keep using the same proxy configured in his browser even when a proxy or gateway that is hosted (and therefore trusted) by a bank is available for online banking. That could be very risky for important information

or transactions. To avoid proxy misuse, when starting a secure session, **C** should first connect to **S** rather than a proxy. That is the order in which MC-SSL proceeds.

The following list summarizes why MC-SSL enhances end-to-end security in the presence of application proxies.

1. **P** is authenticated by **S** as a proxy instead of a client; therefore, **P** cannot impersonate **C**. **P** authenticates itself using its certificate and/or the session id received from **C** as a security token. If session id is a cryptographically random string generated by **S**, **P** does not have to possess a PKI certificate.

2. As described earlier, because **C** first connects and negotiates with **S** rather than **P**, and a proxy is decided after **S** already has the security policy and device capabilities of **C**, the risk of misusing proxies is greatly reduced. **S** can decide if **C** needs a proxy and what proxy is needed.

3. Sensitive data such as id/password and credit card information can be transported through the end-to-end channel, while relatively non-sensitive Web pages, software, and email attachments can go through proxies for content scanning or transforming. Content scanning proxies are expected to help mobile devices enhance system security. Neither an end-to-end SSL connection nor an SSL proxy chain provides both benefits.

4. A proxy channel can be explicitly negotiated as a one-way channel, which eliminates the chance that **P** turns a response channel into a request channel. For instance, we can have a one-way proxy channel that only allows responses from **S** to **C**; all

requests from **C** to **S** have to go through a secure end-to-end channel. As a result, **P** cannot send any fake request to **S** to trigger a sensitive transaction.

5. When unmodifiable data is delivered through **P**, the MAC of the data is calculated and sent to the receiver through the end-to-end channel, and hence **P** cannot modify the data without being detected.

6. When modification is permitted, content attributes such as content types are sent to the receiver through the end-to-end channel. Information of content types can prevent **P** from injecting dangerous code or contents.

There are certainly security issues unsettled in MC-SSL proxy protocol. Since the proxy protocol is on top of SSL, the strength of communication security between two entities is no stronger than that of SSL. This section only discusses the security issues that originate from the complexity of multiple-channel model and the inherent risk of application proxies.

1. It is not easy for **S** and **C** to decide on using which channel to deliver data: end-to-end channel or a proxy channel. There are a number of questions to answer such as the following: What is the sensitivity of the data? Is the data too sensitive to be delivered through the proxy? What should be done if sensitive data needs a proxy? What is the worst consequence if **P** modifies the data? What should be done if insensitive data mixes with sensitive data? For **S**, the security attributes of contents such as sensitivity level need to be defined beforehand. Moreover, **S** needs to have the security policies to help answer those questions. For **C**, it is normally much harder to answer them. A conservative strategy is for **C** to choose the end-to-end channel whenever the answer

is unclear. Fortunately, in typical Internet applications, **S** normally does not have to use a proxy to process the data from **C**. Moreover, it is generally not a problem for **C** to encrypt requests since the traffic volume of requests is usually much less than that of responses from **S**. In brief, **S** and **C** must define their security policies, security attributes of data or contents, and device capabilities. How to define them is beyond the scope of this research.

2. The proxy protocol alone cannot guarantee that **P** has correctly performed its task. When data is modifiable, there are mainly two types of threats: first, **P** can modify requests or responses between **C** and **S**; second, **P** could inject malicious codes if the original content has some embedded code. The strategy to thwart the first threat is neither to send sensitive data through an untrustworthy proxy, nor to use any data from it as sensitive data. To mitigate the second threat, **S** may deliver contents without embedded code, or with embedded code that is safe even if modified.

If some data that require end-to-end protection have to go through a proxy channel, XML Security [13, 14] might be a good solution. One can use different keys to protect different parts of an XML document. For instance, **S** can use the encryption key of the primary end-to-end channel to encrypt an XML element that requires end-to-end confidentiality before **S** wraps the XML document into an APP_DATA_TO_PROXY message, and sends the message to **P**.

5.10.2 Secondary Channels

A typical SSL session uses a cipher suite that includes a 128-bit cipher and a hash algorithm. With MC-SSL, we can negotiate an additional channel with MAC protection, but without

encryption. This channel could be useful for transporting data that does not need confidentiality but require integrity or authenticity. Such a channel combination is analogous to that of IPSec, in which Authentication Header (AH) Protocol and Encapsulating Security Payload (ESP) Protocol can separately provide authenticity or authenticity plus confidentiality.

A combination of block cipher and stream cipher is another sample usage of multiple cipher suites. Applications of streaming media often use a stream cipher for confidentiality or privacy, but a block cipher is probably necessary for user authentication. For SSL, it has to renegotiate its cipher suite or open an additional connection.

Cleartext channels are also available to applications in MC-SSL, but applications must be very cautious with a cleartext channel. As a rule of thumb, if a cleartext channel could possibly compromise the confidentiality or integrity of a communication session, it should not be negotiated at all. In many cases, a channel with MAC protection is preferred. On the other hand, cleartext channels could be very useful. A typical usage is to transport non-sensitive information after a person has been authenticated. For example, a person can choose to read emails through a cleartext channel after his authentication. Another typical usage is to deliver "already-secured" information to eliminate excessive cryptographic protection. Examples of "already-secured" information include but not limited to digitally signed documents or software, and data or documents protected by XML Security or other techniques.

Chapter 6

Implementation and Case Studies

We have developed a prototype system of MC-SSL. The SSL library adopted for this implementation is OpenSSL [37], a popular open source SSL library written in "C" language. The prototype system is developed on Linux platform and written in "C" language. The prototype has been useful for improving the ideas and the protocols of MC-SSL, and for demonstrating that MC-SSL is feasible and functional.

6.1 Message Formats of MC-SSL

According to the protocol designs described in Chapter 5, the prototype has specified the message formats of MC-SSL protocol. They are gathered in Appendix 2 for reference. The messages in Appendix 2 are defined with the same presentation language that is used for TLS 1.0 in RFC 2246 [3].

6.2 State Diagrams

After the initial handshake of MC-SSL, applications at the client and server side have the primary end-to-end channel, through which they can start transporting application data, or negotiating other channels. Figure 6-1 shows the basic state diagram of an MC-SSL session at **C** or **S**. There are four states in the diagram. The exit state is omitted since an MC-SSL session can exit from any state.

Starting from the initial state, the state of an application could transfer to data reading or data writing depending on the actual application protocol. For instance, an HTTP client enters data writing state, and an HTTP server (Web server) enters data reading state. Applications at **C** or **S** can reasonably transfer between data reading and data writing state according to their predefined application protocol, but they cannot both keep reading or writing data. If one of them sends data, the other must read it out; otherwise, a communication deadlock will happen.

In MC-SSL, **C** and **S** can issue a request message to set up and cancel a channel any time when they have the need. However, to avoid communication deadlocks, an application in data reading state should not write a channel request message. Only an application in data writing state has the right to do so. As shown in Figure 6-1, MC-SSL automatically transfers from data reading state to channel control state after receiving a channel request. In contrast, it transfers from data writing state to channel control state when the application above MC-SSL calls an MC-SSL API function to send a channel request. In channel control state, MC-SSL is not supposed to receive or send any application data. When a channel control process is finished, MC-SSL immediately returns to the state prior to channel control state. In Figure 6-1, if MC-SSL enters channel control state by route 3, then it will go back to data reading state by route 4; similarly, it will go back to data writing state if it enters channel control state by route 5.

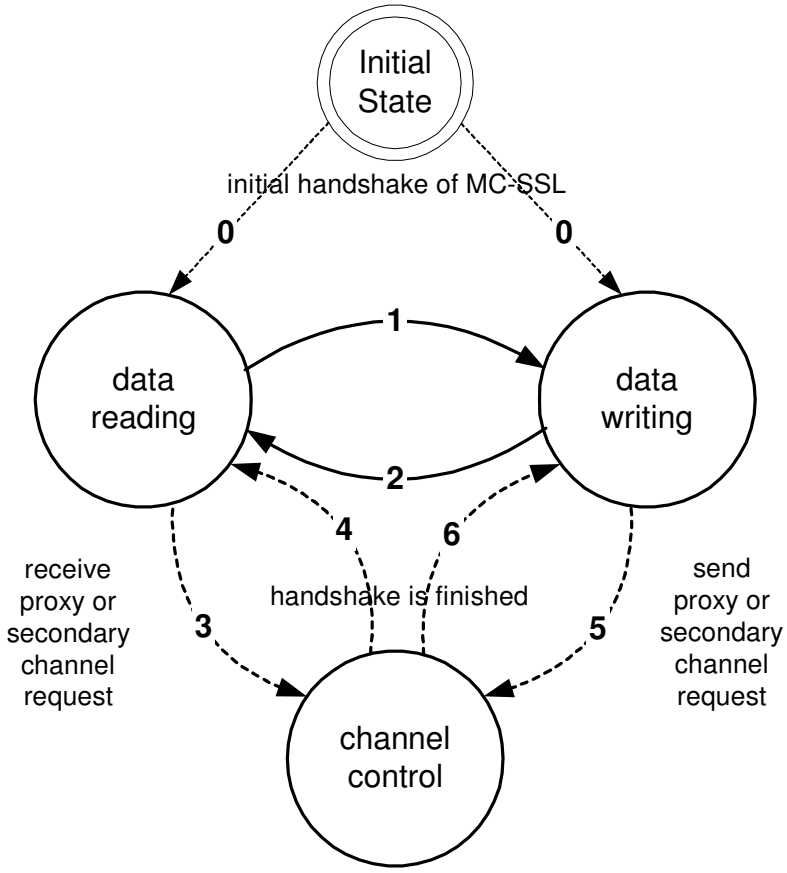

Figure 6-1: Basic State Diagram at a Client or Server

The basic state diagram of MC-SSL at a proxy is illustrated in Figure 6-2. A proxy enters into channel control state from the initial state after receiving a proxy request. If the negotiation of a primary proxy channel succeeds, a proxy goes to the state of data processing and forwarding. Since the job of a proxy is to read data, process it, and forward the result to the next hop or the data receiver, data reading and data writing in Figure 6-1 are combined into a single state in Figure 6-2, namely data processing and forwarding. If a request message for channel control, such as secondary channel negotiation, is received, then MC-SSL at the proxy transfers to channel control state. MC-SSL stays in channel control state until the handshake of channel control is finished.

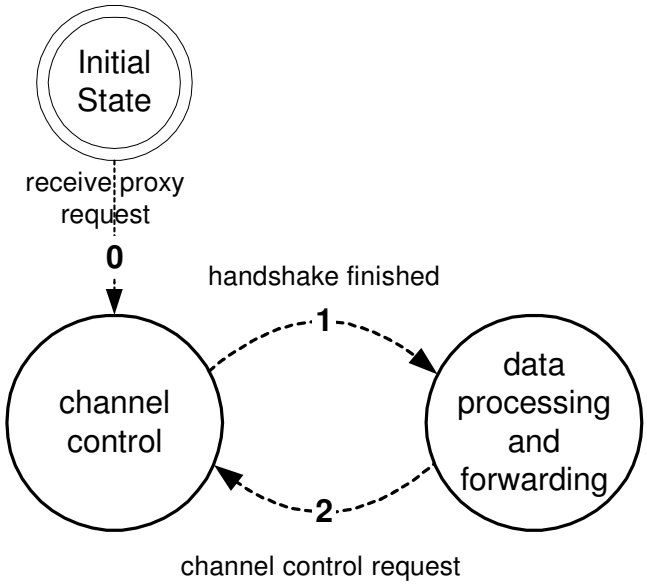

Figure 6-2: Basic State Diagram at a Proxy

6.3 System Diagram of the Prototype

The prototype system has two objectives. One objective is to design and implement the API (Application Programming Interface) library of MC-SSL, which is supposed to be a library suitable for different applications. This thesis does not record the details of MC-SSL API, such as the interface and usage of each API function. The other objective is to test the ideas and protocols of MC-SSL by using MC-SSL to transport Web pages, namely testing HTTP over MC-SSL.

The system diagram of the prototype is shown in Figure 6-3. The server at **S** is composed of two processes: http_server and http_server_p, and they reside in the same computer. The http_server_p is responsible for listening and setting up a connection between http_proxy2 and http_server; therefore, it works like a local proxy at **S**. The http_server process listens on port 5678, and the http_server_p process listens on port 5677. They communicate using a

pair of file pipes on Linux platform. The proxy servers at **P1** and **P2** also listen on 5677 as http_server_p process does. The prototype assigns TCP port 5677 for proxy channels, and TCP port 5678 for end-to-end channels. All server and proxy server processes in the prototype are implemented as concurrent multi-process servers so that they can provide HTTP over MC-SSL services to a number of clients.

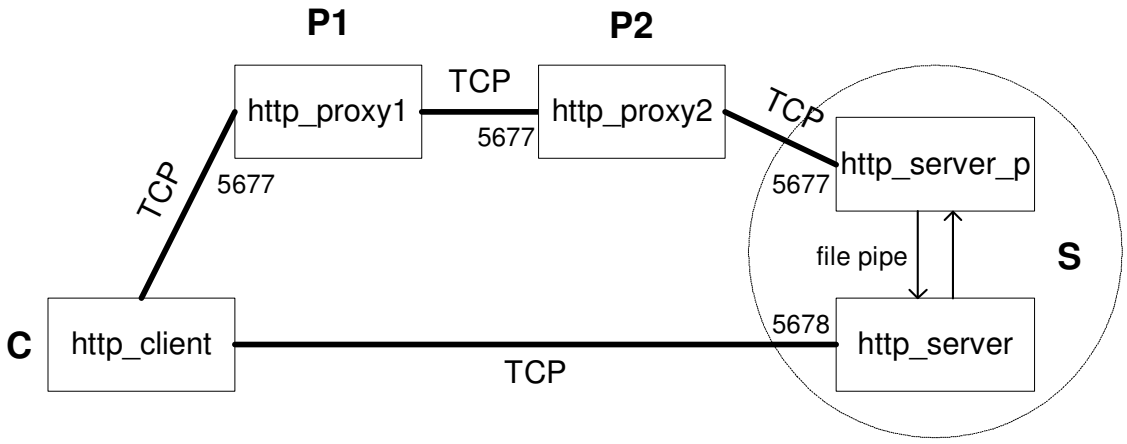

Figure 6-3: System Diagram of the Prototype

The test of the prototype demonstrates the following points: the proxy and secondary channels of MC-SSL can be implemented over SSL and its extension; based on OpenSSL library, MC-SSL protocols can be developed into a straightforward API library for applications. The prototype implementation of MC-SSL can be further extended or simplified according to a specific application scenario. For instance, the **C**-**P1** connection could run over WTLS/WDP protocol stack if both **C** and **P1** supports WAP protocol stack.

6.4 Protocol Names

In the future, if vendors or developers adopt MC-SSL to secure Web sites, we might as well specify the protocol name of HTTP over MC-SSL as “HTTPMS”. For example, the URL of

an HTTPS (HTTP over SSL) web page, such as “https://webmail.ece.ubc.ca”, would become “httpms://webmail.ece.ubc.ca”. A Web browser can connect to an HTTPMS-enabled web site if it supports MC-SSL. If a web browser does not support MC-SSL, it can connect to the equivalent SSL-enabled web site by choosing the alternative HTTPS URL.

Perhaps we can benefit from defining the upper layer of MC-SSL as an independent protocol with the name: MC (Multiple Channel) protocol. MC can provide client-server applications with proxy channels and end-to-end channels. **C** and **S** can use the proxy channel protocol described in Section 5.3 to negotiate single-hop or multi-hop proxy channels between them. Without underlying SSL protocol, MC cannot provide channels with security protection, and secondary channels are not available. The protocol name for HTTP over MC can be specified as “HTTPM”. The relation between HTTPM and HTTPMS is analogous to that between HTTP and HTTPS. Similarly, there would be SMTPM and SMTPMS for SMTP (Simple Mail Transfer Protocol), and POPM and POPMS for POP3 (Post Office Protocol version 3).

6.5 An Application Case Study

In order to show that MC-SSL is practically useful, this section discusses the application of MC-SSL in Web applications. Suppose that we are users of handheld devices, and we want to access some Web servers that require confidentiality, integrity, and/or authentication. Examples of such Web applications include online banking, online shopping, corporate database applications, etc. These Web sites have well designed HTML Web pages, but they are not compatible with Web browsers or micro browsers in handheld devices. Some Web sites have already developed specific WML pages for WAP devices or plainer HTML/

XHTML pages for other handheld devices, but it is very difficult for them to satisfy every type of web browsers and handheld devices. Some Web sites change their Web pages very frequently. It is hard for them to create Web pages for all kinds of devices. They inherently need application proxies to transform Web pages for a given type of handheld devices. However, end-to-end security is a big concern if some information in a Web page is sensitive. At the same time, a large part of a Web page is often non-sensitive. An active handheld user may require a Web server to take away excessive security protection, which reduces the workload of a Web server as well. The solution described below is able to enhance end-to-end security and support selective security for handheld users of Web applications.

Suppose that we would like to use a handheld device to do online banking. In particular, we log into a banking Website, pay a bill, and check recent statements. However, the Web site is not compatible with the browser of the handheld device. We choose a proxy server provided by a wireless telecommunication company to perform transforming. We are not willing to expose password and financial information to the proxy although it is relatively trustworthy. How can MC-SSL address this issue? First, let us consider what channels are required in this scenario. The primary end-to-end channel (Channel 1 in Figure 6-4) is always necessary in an MC-SSL session. Moreover, Channel 1 can be used to protect id/password pair, and other sensitive data including payment information, account number, and bank statements. Channel 3 is a MAC channel without encryption. The hash algorithm could be MD5 or SHA-1. The purpose of Channel 3 is to transport contents that need end-to-end authenticity. To make use of the proxy service, the handheld device must negotiate a single-hop proxy channel (Channel 4 in Figure 6-4) with the Web server. This channel is a simplex

channel that only allows data traffic from the Web server to the handheld device. All HTTP requests generated by the handheld device are sent through Channel 1 because it is hard for a Web browser, which does not know about specific application logic, to judge the sensitivity of data. Channel 4 is also a channel protected only with MAC. Channel 2 is a primary proxy channel, which is used by MC-SSL to set up and manage Channel 4, but it is not employed for transporting application data. Channels 3 and 4 can significantly reduce redundant encryption if they are used in the right way. For example, in a typical Web page for paying a bill, only the information of account number and payees are confidential. Other page content does not have to be encrypted by the Web server. On the other hand, if someone is not concerned about battery life and prefers extra data security, the Web server can simply use Channel 2 without negotiating Channel 4. Moreover, one can always choose not to use an application proxy no matter whether the handheld device can access a server or not.

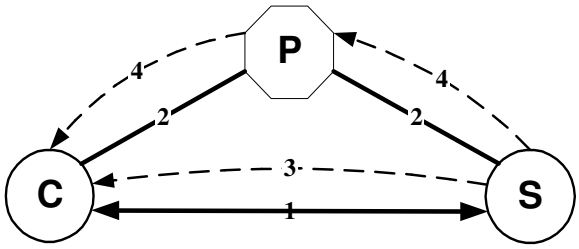

Figure 6-4: Channel Planning for Online Banking

A Web page contains roughly three types of content: the first type is the data that a Web page is created to carry, such as texts, URL links, images, and sound; the second type is the formats of data including HTML tags, fonts, size, colours, etc; the third type is executable code such as JavaScript and Java. The first type can be sensitive or non-sensitive. The second type is relatively non-sensitive. The third type generally requires authenticity and integrity, but does not require confidentiality. Since all HTTP requests go through Channel 1, the

problem is how to send a Web page to **C**. It seems that **S** can simply use Channel 1 to send all the sensitive data, use Channel 3 to send executable codes, and use Channel 4 to send formats of data to **P** for transforming, but how can **C** puts data and codes back to a Web page after a Web page has been changed by **P**.

To solve this new problem, we can use HTML and XML tags and attributes to separate data from its formats and positions in a HTML page. Data can be kept in the same HTML page or be moved to a new URL. HTML attributes such as “datasrc”, “datafld”, and “src” can achieve this objective. The following is an example that separates data in a table from its tabular form.

```
<html> <body>
<xml id="bs_data" src="bs_data.xml"> </xml>
<table border="1" datasrc="#bs_data">
<tr>
        <td> <span datafld="DATE"> </span> </td>
        <td> <span datafld="DETAILS"> </span> </td>
        <td> <span datafld=”DC”> </span> </td>
        <td> <span datafld=”BALANCE”> </span> </td>
</tr>
</table>
</body> </html>
```

The following is bs_data.xml, which is the data source of the table.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ST_DATA>
<TRANS>
        <DATE>2004-09-28</DATE>
        <DETAILS>payroll 23456</DETAILS>
        <DC> 3000.00 </DC>
        <BALANCE>51678.26</BALANCE>
</TRANS>
<TRANS>
        <DATE>2004-10-01</DATE>
        <DETAILS>cheque 00135</DETAILS>
        <DC> -600.00 </DC>
        <BALANCE>51078.26</BALANCE>
</TRANS>
</ST_DATA>
```

In this example, **S** can use Channel 1 to transport the XML file, and use Channel 4 to process the HTML formats. However, “datasrc”, “datafld”, and “src” are not standard HTML attributes even in latest HTML 4.01 [38] although Microsoft Internet Explorer supports these attributes. Fortunately, XHTML (Extensible Hypertext Markup Language) [39], the successor of HTML, has defined embedding attributes: “src=URI” and “type=ContentTypes”. These two attributes are used to embed content from other resources

into the current element. The “src” attribute specifies the location of a source for the contents of the element, and the “type” attribute specifies the allowable content types of the relevant URI. The following are two examples:

```
<div  src="bs_data.xml"  type="application/xml"> </div>

<script  src="popwin"  type="application/x-javascript"/>
```

By using embedding objects, files, or data, a Web page can be divided into a various number of parts for different channels. However, if the proxy **P** is hacked by a malicious person, he could modify “src” or “datasrc” attribute, and thus a user could see fake data. For the following reasons, this is only a small risk: first, Web browsers will not use URL links that point to a different Web site in a Web page protected by SSL or MC-SSL. Second, **P** cannot get confidential data because all of them go through the end-to-end channel. Third, **C** or **S** should choose a relatively trustworthy proxy to reduce this risk. In our example of online banking, a proxy server provided by a telecommunication company should be good enough although a proxy server of a bank association is even better if that is available. There are still some methods to minimize the risk. For example, **S** can collect all URI/URL in a Web page and send a copy to **C** through Channel 1 or 3. Alternatively, **S** can send the hash value of all URI/URL to **C**. This kind of methods is helpful if they are adopted by HTML or XHTML standards.

The benefit of selective security is also demonstrated by using the channel planning illustrated in Figure 6-4. Channel 3 and 4 are used for non-confidential data and Channel 1 for confidential data. Suppose that Channel 1 uses 128-bit AES for encryption and MD5 for MAC, and Channel 3 and Channel 4 use MD5 for MAC protection. If 95 percent of a Web page is non-confidential, 71% of the CPU time can be saved by Channel 3 and 4. If the non-

confidential part is 80%, then MC-SSL can save 57% of the CPU time that is spent on cryptographic operations. In many cases, non-confidential information could contribute to more than 95% of a Web page secured by SSL. Depending on what algorithms are negotiated for data encryption and MAC protection, MC-SSL channels can often save 45% to 90% of the CPU time spent on cryptographic operations.

Chapter 7

Conclusion and Future Work

This thesis proposes Multiple-Channel SSL, a new security model and protocol extended from SSL. MC-SSL has three main features: first, it improves end-to-end security in the presence of partially trusted application proxies; second, MC-SSL supports secondary channels and channel direction restriction so that appropriate communication security can be selectively applied to different data or content; third, MC-SSL supports channel negotiation according to security policies, device capabilities, and security attributes of content. The multiple-channel model of MC-SSL is more flexible than SSL, and hence it is able to satisfy diverse requirements in different application scenarios, especially for emerging mobile applications.

The MC-SSL protocol presented in this thesis is only one possible implementation of MC-SSL model. The principle and model of MC-SSL can be applied to improve WTLS protocol or develop a counterpart protocol of MC-SSL for UDP applications. We can develop a similar security protocol on top of UDP so that applications such as VoIP can make use of proxy channels and multiple cipher suites. If two wireless terminals communicate with VoIP over RTP, but they do not support the same voice coding or compression scheme, they can use MC-SSL to set up a proxy to translate the voice coding. In addition, they can use different cipher suites for user authentication and voice traffic as mentioned in Section 5.9.2. Another interesting direction is about how to combine MC-SSL with XML to further improve end-to-end security and support selective security. As shown in

the application case of Section 6.5, we can solve practical problems of client-server applications by combining MC-SSL with XML-based methods. One can even use XML and XML Security to develop a security protocol at application layer, which reflects the idea of MC-SSL and has the flexibility of XML.

For current MC-SSL protocol, we need to further analyze and enhance its security. Since MC-SSL is a new security protocol that is more complex than SSL, it is almost certain to have new vulnerabilities. We can gradually eliminate them by means of informal analysis or formal verification. Since inappropriate usage of application proxies and cipher suites impairs communication security, application developers must understand how to negotiate and use different channels in the right way. Therefore, practical guidelines, security policies, and case studies for different applications are indispensable in order to put MC-SSL into use. Besides, the design of user interfaces for configuring MC-SSL is important for application users to get the security protection they want. In terms of implementation and tests, it is also important to develop MC-SSL for handheld devices, and do comparative tests.

Bibliography

[1] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, Jan. 1999

[2] B. Schneier, *Applied Cryptography*, *2nd Edition*, John Wiley & Sons, 1996.

[3] S. Ravi, A. Raghunathan, and N. Potlapally, "Securing Wireless Data: System Architecture Challenges," *Proc. Int'l Symp. System Synthesis,* pp. 195-200, Oct. 2002.

[4] M. Portmann and A. Seneviratne, "Selective Security for TLS," *Proc. IEEE 9th Int'l Conf. on Networks*, pp. 216-221, Oct. 2001.

[5] WAP Forum, WAP 2.0 Specifications, http://www.openmobilealliance.org/.

[6] Y. Song, V. C.M. Leung, and K. Beznosov, "Supporting End-to-end Security across Proxies with Multiple-Channel SSL", *Proc. 19th IFIP International Information Security Conf.*, pp. 323-337, Aug. 2004.

[7] Y. Song, V. C.M. Leung, and K. Beznosov, "Implementing Multiple Channels over SSL", *Proc. 1st Int. Conf. on E-Business and Telecommunication Networks*, Setúbal, Portugal, Aug. 2004.

[8] M.Y. Rhee, *Internet Security: Cryptographic principles, algorithms, and protocols*, John Wiley & Sons, 2003.

[9] GSM Association, GPRS, http://www.gsmworld.com/technology/gprs.

[10] T-Mobile, T-Mobile Internet Accelerator, http://www.t-mobile.com/help/services/tz /accel.asp.

[11] Palm Source, Inc., Web Clipping Applications Development, http://www.palmos.com /dev/tech/webclipping.

[12] Symantec Corp., SymbOS.Cabir, http://securityresponse.symantec.com/avcenter/venc/data/epoc.cabir.html.

[13] W3C, XML Signature Recommendations, http://www.w3.org/Signature/, Feb. 2002.

[14] W3C, XML Encryption Recommendations, http://www.w3.org/Encryption/, Dec. 2002.

[15] E.K. Kwon, Y.G. Cho, and K.J. Chae, "Integrated Transport Layer Security: End-to-End Security Model between WTLS and TLS," *Proc. IEEE 15th Int'l Conf. on Information Networking*, pp. 65-71, Jan. 2001.

[16] D. J. Kennedy, "An Architecture for Secure, Client-Driven Deployment of Application-Specific Proxies," Master Thesis, University of Waterloo, 2000.

[17] JupiterMedia Corp., "HTTP Compression Speeds up the Web," http://webreference.com/internet/software/servers/http/compression.

[18] V. Gupta, S. Gupta, S. Chang, and D. Stebila, "Performance Analysis of Elliptic Curve Cryptography for SSL," *Proc. ACM workshop on Wireless security*, pp. 87-94, Sep. 2002.

[19] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana, "Optimizing Public-Key Encryption for Wireless Clients," *Proc. IEEE Int'l Conf. on Communications*, pp. 1050-1056, May 2002.

[20] R.B. Lee, Z. Shi, and X. Yang, "Efficient Permutations for Fast Software Cryptography," IEEE Micro, Vol. 21, pp. 56-69, Dec. 2001.

[21] D. Boneh and N. Daswani, "Experimenting with Electronic Commerce on the Palm Pilot," *Proc. 3rd Int'l Conf. on Financial Cryptography*, pp. 1-16, Feb. 1999.

[22] J. Burke, J. McDonald, and T. Austin, "Architectural Support for Fast Symmetric-Key Cryptography," *Proc. 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 178-189, Nov. 2000.

[23] IBM Corp. and Microsoft Corp., "Security in a Web Services World: A Proposed Architecture and Roadmap," http://www-106.ibm.com/developerworks/webservices /library/ws-secmap/, Apr. 2002.

[24] OASIS Open, "Web Services Security: SOAP Message Security," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss, Aug. 2003.

[25] M. Naedele, "Standards for XML and Web services security," *IEEE Computer*, pp. 96-98, Apr. 2003.

[26] OASIS Open, "Web Services Security X.509 Certificate Token Profile," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss, working draft 11, Oct. 2003.

[27] OASIS Open, "Web Services Security Kerberos Certificate Token Profile," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss, working draft 03, Jan. 2003.

[28] OASIS Open, "Web Services Security Username Token Profile," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss, working draft 04, Oct. 2003.

[29] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport Layer Security (TLS) Extensions," RFC 3546, 2003.

[30] S. Bellovin, "Problem Areas for the IP Security Protocols," *Proc. 6th USENIX Security Symposium*, pp. 205-214, 1996.

[31] S. Bellovin, "Probable Plaintext Cryptanalysis of the IP Security Protocols," *Proc. of the Symp. on Network and Distributed System Security*, pp. 155-160, Feb. 1997.

[32] D. Wagner and Bruce Schneier, "Analysis of the SSL 3.0 Protocol," *Proc. 2nd USENIX Workshop on Electronic Commerce*, pp. 29-40, Nov. 1996.

[33] J. C. Mitchell, V. Shmatikov, U. Stern, "Finite-state Analysis of SSL 3.0 and Related Protocols," *Proc. 7th USENIX Security Symposium*, pp. 201-216, Jan. 1998.

[34] L. C. Paulson, "Inductive Analysis of the Internet Protocol TLS," *ACM Trans. on Information and System Security*, Vol. 2, No. 3, pp. 332–351, Aug. 1999.

[35] D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on RSA Encryption Standard PKCS #1," *Proc. Advances in Cryptology -- Crypto' 98*, pp. 1-12, 1998.

[36] M. Saarinen, "Attacks against the WAP WTLS Protocol," *Proc. of IFIP Communications and Multimedia Security 1999*, pp. 209-215, Sep. 1999.

[37] OpenSSL Project, http://www.openssl.org/, 2004.

[38] W3C, HTML 4.01, http://www.w3.org/TR/html4/, Dec. 1999.

[39] W3C, XHTML 2.0, http://www.w3.org/TR/xhtml2/, Jul. 2004.

[40] V. Gupta and S. Gupta, "Experiments in Wireless Internet security," *Proc. IEEE Wireless Communications and Networking Conference 2002*, Vol. 2, pp. 860-864, Mar. 2002.

[41] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley, 2001.

[42] R. Smith, *Authentication: from Passwords to Public Keys*, Addison-Wesley, 2002.

Appendix A

Overview of SSL

This part attempts to give an overview of SSL protocol, and it is extracted from the publication of V. Gupta and S. Gupta [40].

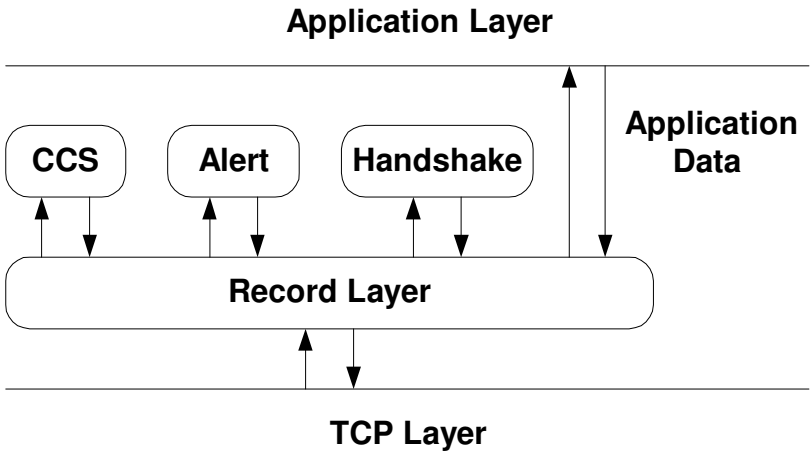

Figure A-1: SSL Architecture [40]

SSL offers encryption, authentication, and integrity protection of application data over insecure, public networks. Figure A-1 shows the layered nature of this protocol. The Record layer, which sits above a reliable transport service like TCP, provides bulk encryption and authentication using symmetric key algorithms. The keys for these algorithms are derived from a master secret established by the Handshake protocol between the SSL client and server using public-key algorithms. SSL is very flexible and can accommodate a variety of algorithms for key agreement, encryption and hashing. To guard against adverse interactions (from a security perspective) between arbitrary combinations of these algorithms, the

standard specification explicitly lists combinations of these algorithms, called cipher suites, with well-understood security properties. The Handshake protocol is the most complex part of SSL with many possible variations (Figure A-2). In the following subsection, we focus on its most popular form, which uses RSA key exchange and does not involve client-side authentication. While SSL allows both client- and server-side authentication, only the server is typically authenticated due to the difficulty of managing client-side certificates. Client authentication, in such cases, happens at the application layer, e.g. with passwords sent over the SSL-protected channel.

1) Full SSL Handshake: When an SSL client encounters a new server for the first time, it engages in the full handshake shown in Figure A-2 (a). The client and server exchange random nonce (used for replay protection) and negotiate a mutually acceptable cipher suite in the first two messages. The server then sends its RSA public-key inside an X.509 certificate. The client verifies the public-key, generates a 48-byte random number (the pre-master secret) and sends it encrypted with the server's public-key. The server uses its RSA private-key to decrypt the pre-master secret. Both end-points use the pre-master secret to create a master secret which, along with previously exchanged nonce, is used to derive the cipher keys, initialization vectors, and MAC (Message Authentication Code) keys for the Record Layer.

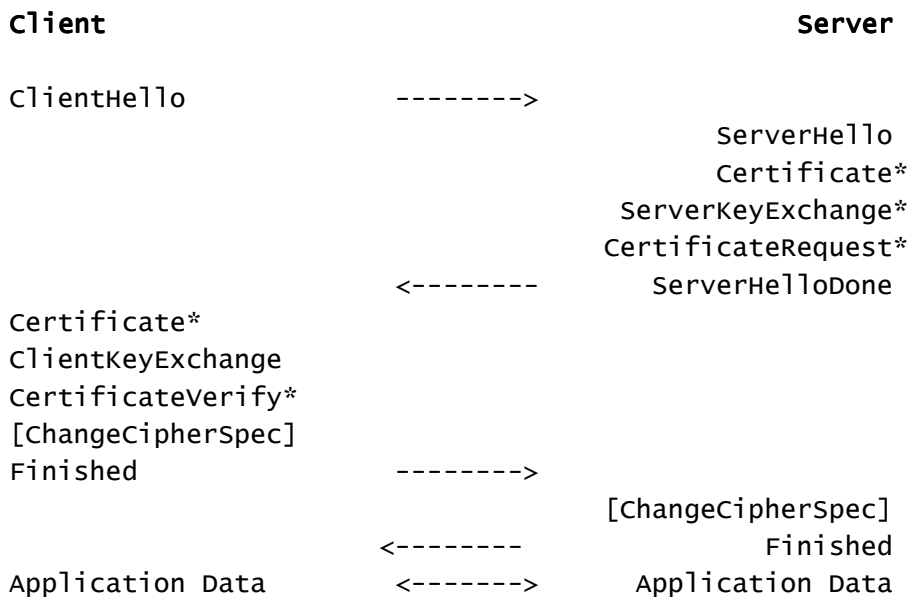

Figure A-2 (a): Full Handshake of SSL

2) Abbreviated SSL Handshake: A client can propose to reuse the master key derived in a previous session by including that session's (non-zero) ID in the first message. The server indicates its acceptance by echoing that ID in the second message. This results in an abbreviated handshake without certificates or public-key cryptographic operations so fewer (and shorter) messages are exchanged (see Figure A-2 (b)). An abbreviated handshake is significantly faster than a full handshake.

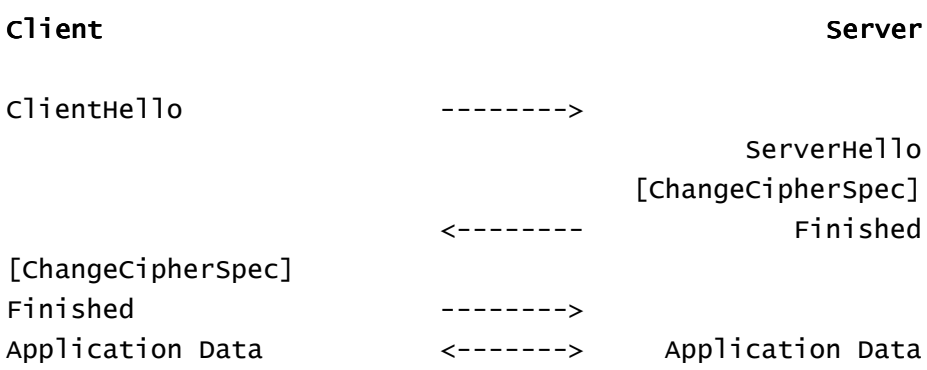

Figure A-2 (b): Abbreviated Handshake of SSL

Appendix B

Message Formats of MC-SSL Protocol

The following example is given by RFC 2246 [1] to help understand the presentation language for message formats. This example shows the syntax to define data types such as enumeration, structure, and fixed and variable length vector. VariantMessage is a structure with a variable body. Uint8, uint16, uint32, and opaque are built-in data types.

```
enum { apple, orange } Fruit;          /* enumeration */
struct {
   uint8 one_byte;                     /* uint8: 1-byte number */
   uint16 number;                      /* uint16: 2-byte number */
   opaque string<0..10>;               /* variable length vector: 0 to 10 bytes */
} V1;
struct {
   uint32 number;                      /* uint32: 4-byte number */
   opaque string[10];                  /* fixed length vector: 10 bytes */
} V2;
struct {
   select (Fruit) {                    /* value of selector is implicit */
      case apple: V1;                  /* variant body, type = apple */
      case orange: V2;                 /* variant body, type = orange */
   } variant_body;
} VariantMessage;                     /* a message with a variable body */
```

The rest of this appendix presents the message formats of MC-SSL protocol.

1. Basic message elements

```
enum {
    client_hello (1), server_hello (2), client_security_policy (3), client_capabilities (4),
    proxy_suggestion_s2c (5), proxy_request_c2s (6), proxy_request_response_s2c (7),
    proxy_request_c2p (8), client_authen_req_p2c (9), client_authen_resp_c2p (10),
    proxy_request_p2p (11), proxy_request_p2s (12), proxy_finish (13), cp_verification
    (14), cp_veri_finish (15), app_data_direct (16), app_data_from_proxy (17),
    app_data_to_proxy (18), app_data_control_proxy (19), sec_chan_req (20),
    sec_chan_resp (21), chan_cancel_req (22), chan_cancel_resp (23)
} MessageType;

opaque SessionID <0..32>;
uint8 CipherSuite [2];
enum { SHA-1 (1), MD5 (2) } HashAlgorithm;
enum { DUPLEX (0), C2S (1), S2C (2), NONE (3) } ChannelDirection;
enum { Full (0), Abbreviated (1) } HandshakeType;
enum { No (0), Yes (1) } YesOrNo;
enum {unchange_bit (1), modify_bit (2), discard_bit (4)} ChangeBitMask;

struct {
    uint32 gmt_unix_time;
```

```
        opaque random_bytes [28];
    } Random;

    struct {
        uint8 major_ver;
        uint8 minor_ver;
    } ProtocolVersion;
    struct {
        HashAlgorithm hash;
        opaque MAC_write_secret [16];
    } MACParameters;
```

2. Message header

```
    struct {
        MessageType msg_type;       /* message type */
        uint32 length;              /* total length of message body */
    } MSG_HEADER;
```

3. Initial Handshake Protocol

```
    struct {
        MSG_HEADER header;
        SessionID sid;
```

```
    ProtocolVersion ver;
    MACParameters e2e_mac;     /* parameters for end-to-end MAC */
} HELLO;

HELLO     CLIENT_HELLO;
HELLO     SERVER_HELLO;

struct {
    MSG_HEADER header;
    opaque policy[header.length];
} CLIENT_SECURITY_POLICY;

struct {
    MSG_HEADER header;
    opaque capabilities[header.length];
} CLIENT_CAPABILITIES;
```

4. Primary Proxy Channel Protocol

```
struct {
    opaque proxy;                  /* DNS name or IP address of a proxy */
    uint8 port;                    /* proxy port number */
    opaque services <0..256>;      /* required proxy services (separated by semicolon) */
    opaque certificate <0..2^24-1>;        /* certificate or its URL of a proxy */
```

```
} ProxyParameters;

struct {
    MSG_HEADER header;
    uint8 cid;
    ChannelDirection chan_dir;
    ProxyParameters proxy_list <0..32>;          /* maximum 32 proxies in a channel */
} PROXY_REQUEST;

PROXY_REQUEST       PROXY_SUGGESTION_S2C;
PROXY_REQUEST       PROXY_REQUEST_C2S;

struct {
    MSG_HEADER header;
    YesOrNo answer;                          /* server's answer to client's proxy request */
    opaque reject_reason <0..128>;
} PROXY_REQUEST_RESPONSE_S2C;

struct {
    MSG_HEADER header;
    ProtocolVersion ver;
    SessionID sid;
    uint8 cid;
```

```
    ChannelDirection chan_dir;
    HandshakeType handshake;                    /* full or abbreviated handshake */
    ProxyParameters proxy_list <0..32>;         /* the last proxy is actually the server */
    /* a list of acceptable authentication methods in the order of preference */
    opaque authen_methods <0..256>;
    Random random_list <0..32>;
    YesOrNo verify_client;              /* whether proxies verify C or not */
    YesOrNo verify_proxy;               /* whether C or S verifies proxies or not */
} PROXY_REQUEST_C2P;

struct {
    MSG_HEADER header;
    ProtocolVersion ver;
    opaque authen_method_req <0..256>;  /* authentication required by the first proxy */
    /* whether a client capabilities message will follow or not */
    YesOrNo message_follow;
} CLIENT_AUTHEN_REQ_P2C;

struct {
    MSG_HEADER header;
    uint8 authen_meth;
    opaque authen_data <0..2^24-1>;             /* authentication data */
} CLIENT_AUTHEN_RESP_C2P;
```

```
PROXY_REQUEST_C2P          PROXY_REQUEST_P2P;
PROXY_REQUEST_C2P          PROXY_REQUEST_P2S;

struct {
    MSG_HEADER header;
    YesOrNo result;        /* result of first round negotiation of a proxy channel */
    opaque reject_reason <0..128>;        /* the reason of failure */
    Random random_list <0..32>;
    YesOrNo verify_client;                /* whether proxies verify C or not */
    YesOrNo verify_proxy;                 /* whether C or S verifies proxies or not */
} PROXY_FINISH;

struct {
    MSG_HEADER header;
    Random random_list <0.. 32>;          /* the final random string */
    Signature signature_list <0..32>;
    YesOrNo verify_result_list <0..32>;  /* a list of verification result */
} CP_VERIFICAION;

CP_VERIFICATION      CP_VERI_FINISH;
```

5. Application Data Protocol

```
struct {
    MSG_HEADER header;
    uint16 seq;                         /* sequence number */
    uint8 data <1..2^32-1>;
} APP_DATA_DIRECT;

struct {
    MSG_HEADER header;
    uint16 seq;
    uint8 change_restriction;           /* bit OR of ChangeBitMask */
    opaque service_request <0..1024>;   /* request for proxies */
    opaque content <0..2^32-1>;         /* data or content */
} APP_DATA_TO_PROXY;

struct {
    MSG_HEADER header;
    uint16 seq;
    uint8 change_status;            /* bit OR of ChangeBitMask */
    YesOrNo result;                 /* success or failure of processing */
    opaque content <0..2^32-1>;
} APP_DATA_FROM_PROXY;
```

```
struct {
    MSG_HEADER header;
    uint16 seq;
    uint8 proxy_cid;
    uint8 change_restriction;
    opaque content_attributes <0..1024>;
    opaque MAC <0..256>;        /* the field length depends on the hash algorithm */
} APP_DATA_CONTROL_PROXY;
```

6. Secondary Channel Protocol

```
struct {
    uint8 cid;
    uint8 col_cid;  /* collaborative end-to-end channel of a proxy channel */
    CipherSuite cipher_suites <1..2^16-1>;        /* preferred cipher suites */
    ChannelDirection direction;
} SecondaryChannelReq;

struct {
    MSG_HEADER header;
    SecondaryChannelReq requests <1.. 64>;
    opaque MAC <0..256>;        /* the field length depends on the hash algorithm */
} SEC_CHAN_REQ;
```

```
struct {
    uint8 cid;
    YesOrNo result;          /* OK or not */
    uint8 col_cid;           /* collaborative end-to-end channel */
    CipherSuite cipher_suite;
    opaque MAC <0..256>;     /* the field length depends on the hash algorithm */
} SecondaryChannelResp;

struct {
    MSG_HEADER header;
    SecondaryChannelResp responses <1..64>;
} SEC_CHAN_RESP;
```

7. Channel Cancellation Protocol

```
struct {
    MSG_HEADER header;
    opaque cid_list <0..64>;
    opaque MAC <0..256>;
} CHAN_CANCEL_REQ;

struct {
    MSG_HEADER header;
    opaque cid_list <0..64>;          /* channels to be cancelled */
```

```
        opaque MAC <0..256>;
    } CHAN_CANCEL_RESP;
```

8. Alert Protocol

```
    enum { warning (1), fatal (2), (255) } AlertLevel;
    enum {
            close_notify (0),
            unexpected_message (10),
            message_loss (11),
            message_repeat (12),
            message_timeout (13),
            bad_mac (20),
            corrupted_message (25),
            SSL_handshake_failure (40),
            initial_handshake_failure (41),
            protocol_version (42),
            security_policy_failure (43),
            authentication_failure (50),
            bad_certificate (51),
            unsupported_certificate (52),
            certificate_revoked_or_expired (53),
            illegal_parameter (54),
```

```
        unknown_ca (55),
        unsupported_cipher_suites (60),
        insufficient_security (65),
        nonexistent_channel (70),
        restricted_channel (71),
        internal_error (80),
        user_cancelled (90),
       (255)
} AlertDescription;

struct {
        MSG_HEADER header;
        AlertLevel level;
        AlertDescription description;
       opaque MAC <0..256>;       /* the field length depends on the hash algorithm */
} Alert;
```