IMPLEMENTING MULTIPLE CHANNELS OVER SSL

Yong Song, Victor C.M. Leung, Konstantin Beznosov
Department of Electrical and Computer Engineering,University of British Columbia, Vancouver,Canada
Email:{yongs,vleung,beznosov}@ece.ubc.ca

Keywords: Communication security, Mobile security, Multiple channels, SSL

Abstract: Multiple-Channel SSL (MC-SSL) is our model and protocol for the security of client-server communication. In contrast to SSL, MC-SSL can securely provide applications with multiple channels, and each of them can have a specific cipher suite and a various number of application proxies; meanwhile, the channel negotiation and operation in MC-SSL are still based on SSL, which needs a small change in order to support multiple cipher suites. In this paper, we first introduce the multiple-channel model of MC-SSL, and then focus on the design and implementation of multiple channels over SSL, especially multi-hop proxy channels and secondary channels.

1 INTRODUCTION

To address several limitations of TLS/SSL (Dierks, 1999) (referred as SSL in this paper), we have proposed multiple-channel SSL (MC-SSL) (Song, 2004). Based on vanilla SSL, MC-SSL has several advantages over it. First, MC-SSL supports a various number of application proxies (or gateways) between a client and a server. Second, MC-SSL supports multiple cipher suites in a single connection so that client and server can negotiate multiple cipher suites for different data or contents. Third, new factors such as security policies, device capabilities, and security attributes of data are taken into account in the security model of MC-SSL. As a result, the multiple-channel nature of MC-SSL enables MC-SSL to flexibly meet diverse security requirements from different terminals, servers, applications, and users. In particular, MC-SSL can help resource-constrained devices such as PDAs and cellular phones because they may need application proxies for proxy services such as content transformation or virus scanning, and also they can save battery power and CPU time by using multiple cipher suites.

MC-SSL supports two types of channels between a client and a server: end-to-end and proxy channels. The proxy channel protocol is described (Song, 2004). This paper reports on the next step in this work, design and implementation of multiple cipher suites as well as an extension of single-hop proxy channels to multi-hop proxy channels. The prototype implementation demonstrates that the design of MC-SSL protocol is feasible.

The rest of this paper is organized as follows. Section 2 analyses the functional limitations of SSL that motivated this work. Section 3 describes MC-SSL. Section 4 discusses related work. Section 5 focuses on the design and implementation of MC-SSL protocol. Section 6 concludes the paper.

2 PROBLEM MOTIVATION

SSL is a *de facto* security protocol at transport layer, but it has some functional limitations. First, while SSL can provide a secure point-to-point connection, it does not securely support application proxies. If a proxy **P** is involved between a client **C** and a server **S**, **C** would normally set up an SSL connection with **P**, and then **P** would act as the delegate of **C** and set up another SSL connection with **S**. The purpose of proxies could be virus scanning, content transformation, or compression. An example of such an SSL chain proxy model is the WAP gateway architecture, in which the connection between **C** and **P** is over WTLS, a variant of TLS protocol. The SSL chain proxy model is shown in the lower part of Figure 1, in which there is a various number of proxies from $\mathbf{P}_1$ to $\mathbf{P}_n$. Since any proxy in the chain can read and modify sensitive data at will, this model assume unconditional trust in all proxies at least from one side of the connection. This can be satisfied only if

all proxies are administrated by the organization or individual that also administrates **C** (or **S**). In other cases, **C** (or **S**) has to take risks of information leakage and tampering unless the exchanged information is non-sensitive.

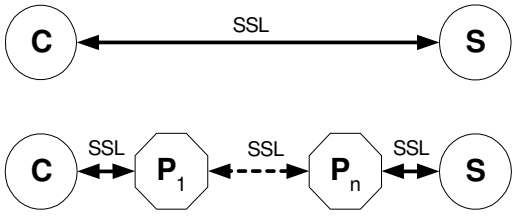

Figure 1. SSL end-to-end model and chain proxy model

The second limitation of SSL is that it can employ only one cipher suite in a connection. Although SSL allows re-negotiating the cipher suite of a connection, frequent re-negotiations are inefficient because of the message interaction and certificate verification in the handshake protocol. Besides, SSL only provides a duplex channel in which the ciphers suites in both directions must be the same at any time. If requests and responses need different protection, then a client and a server have to change cipher suite in every round, which is inefficient for most applications. Accordingly, lots of data is overly protected. For instance, a handheld user checks email all day. She wants encryption for the id/password of her email account when sending requests but she does not require extra confidentiality for her emails. It is good enough for the user to have reasonable level of confidentiality protection for the id/password, but no encryption for email contents. This way, the battery power is preserved. To summarize, the requirement for communication security does not entail the strongest cipher; moreover, security is tightly related to other requirements. The SSL's support for one cipher suite at a time combined with the relatively high cost of changing 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.

In addition, to allow **S** to take various factors into account and optimize the combination of different channels, **C** may want to send **S** its terminal capabilities and security policy. For example, **C** may define whether proxies are allowed to process data with sensitivity below a certain level, and what cipher suites are strong enough to protect data with a certain level of sensitivity. Lack of negotiation support for proxies and multiple cipher suites is the third limitation of SSL. These functional limitations form a mismatch gap between the capabilities of SSL and the requirements of the applications with performance, power, and other constrains. When mobile applications become more popular, the gap will become more apparent. The intent of MC-SSL design is to narrow this gap.

3 HIGH LEVEL DESCRIPTION OF MC-SSL

Three key features of MC-SSL help us to address the above limitations of SSL. Going in reverse order, MC-SSL supports channel negotiation according to the parties' security policies, device capabilities, and security attributes of data. To address second limitation (only one cipher suite at a time and for both directions), MC-SSL supports multiple cipher suites and simplex channels with a specified traffic direction. 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.

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. We can define it as a structure as follows:

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

A 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 become an attribute of a connection. Every MC-SSL connection must first negotiate a cipher suite strong enough 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 in a connection, and it is established using the unchanged SSL protocol. Other channels in an MC-SSL connection are referred as secondary. They are new channels added to an SSL connection so as to support multiple cipher suites per connection. The sample connection shown in Figure 2 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. The protocol to support multiple cipher suites is described and discussed in Section 5.2. Secondary channels are further divided into end-to-end and proxy ones.

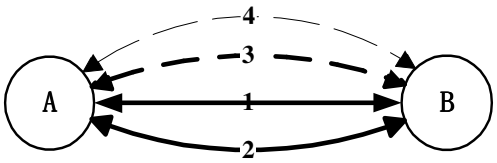

Figure 2. Multiple cipher suites inside a connection

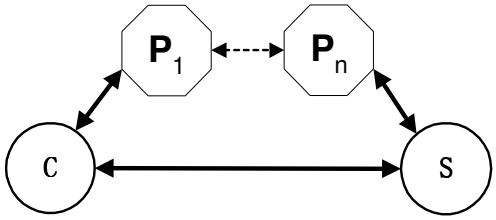

Figure 3. Proxy channel model of MC-SSL

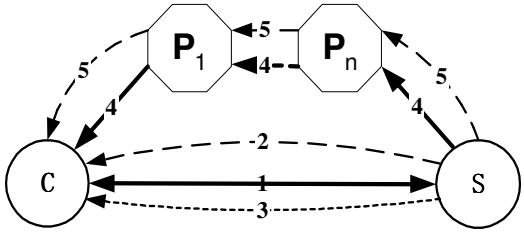

Figure 4. Multiple-channel model of MC-SSL

Figure 3 shows the proxy model of MC-SSL, in which point-to-point SSL connections collectively form a shape of arc. **C**-**S** is an end-to-end channel, and **C**-$\mathbf{P}_1$- … -$\mathbf{P}_n$-**S** is a proxy channel. In this model, **C**-$\mathbf{P}_1$- … -$\mathbf{P}_n$-**S** is no longer an independent chain proxy channel, as in Figure 1. Instead, it relies on the **C**-**S** channel to do channel negotiation and application data transportation. Besides, **C** or **S** can deliberately choose one of the two channels to transport data according to the protection requirements of data so that sensitive information, such as passwords and credit card numbers, do not have to be disclosed to a third-party proxy. An MC-SSL session can have zero or more proxy channels. Each of them and the corresponding end-to-end channel follow the proxy model illustrated in Figure 3. The protocol to negotiate a single-hop proxy channel has been described in (Song, 2004). Section 5.1.2 of this paper discusses the multi-hop proxy channel protocol.

A combination of the proxy model and the multiple cipher suites is illustrated in Figure 4. In MC-SSL, a channel is defined as a communication "pipe" with or without intermediate application proxies. Two MC-SSL endpoints communicate with each other through the pipe using a cipher suite. In addition, a channel can be either duplex, or simplex with a flow direction. An MC-SSL channel is 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 identifier of a channel in an MC-SSL session context. 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. If a channel has no proxies, then it is an end-to-end channel. Direction, *D*, indicates whether a channel is a duplex channel, or a simplex channel pointing to one of the two endpoints. The sample MC-SSL session in Figure 4 has five channels. Among them, channel 1 and 4 are primary channels, and others are secondary channels. 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**. For the client application, **C** uses channel 1 to send encrypted requests to **S**, and **S** may choose one of the five channels to send back responses.

4 RELATED WORK

To avoid information leakage in the case of SSL proxy chain, one approach is to use encryption-based tunnelling to make data unreadable to a proxy. For instance, Kwon et al. (Kwon, 2001) require **C** to encrypt data twice: first for **S** using K_S**,** and then for a proxy using K_P. Consequently, the proxy cannot perform functions such as content transformation and/or scanning. Another approach is to simultaneously have an end-to-end SSL connection and an SSL chain between **C** and **S**, both shown in Figure 1. To provide confidentiality, sensitive data is sent through the end-to-end connection. This is a typical approach, employed, for instance, by (Kennedy, 2000), but it is insecure because proxies impersonate **C** to interact with **S**. Most servers today

still authenticate their clients using id/password. If a proxy keeps a client's id/password, then it can impersonate **C** in unconstrained fashion. There are a number of solutions for **C** to avoid exposing id/password, such as sharing the master key or the symmetric session keys with proxies, or helping a proxy sign the verification data. However, a proxy can still impersonate **C** during a particular session and conduct person-in-the-middle attacks. MC-SSL is securer than this approach simply because every proxy is authenticated with its genuine identity; therefore, there is no impersonation in MC-SSL.

Portmann and Seneviratne proposed a simple extension to SSL to get an extra cleartext channel in an SSL connection (Portmann, 2001). However, the security strength of this method is questionable because the cleartext channel is granted without negotiation and operated without control. Moreover, this method is not capable of creating other types of channels.

Finally, we would like to compare MC-SSL with XML security solutions including XML Security (W3C, 2002) and Web Services Security (WSS) (IBM, 2002; OASIS, 2003). XML Security is a set of core specifications that define XML syntaxes to represent encryption, hash, and digital signature. WSS is a framework that unites a number of existing and emerging specifications for the purpose of constructing comprehensive security solutions for XML-based Web services. WSS uses XML Security as a building block. Compared with XML Security and Web Services Security, MC-SSL is a complete and compact protocol under application layer and is able to provide authentication, key exchange, and secure data transportation for client-server applications with or without proxies. On the other hand, both XML Security and WSS are not self-contained protocols, and they do not attempt to specify a fixed security protocol for authentication and key exchange so that they can have the extensibility and flexibilities to integrate existing or new security technologies at different layers. As with SSL, MC-SSL can be combined with XML Security, or adopted by WSS for securing Web Services. For example, by combining XML Security with MC-SSL, an application can use MC-SSL to do authentication and key exchange for client, server, and proxies, and use XML Security to perform complex encryptions and/or digital signatures on application data. Similarly to SSL, MC-SSL appears to be a more efficient solution than WSS/XML Security for those applications that require bulk protection of data because it avoids key negotiation/generation for each individual message.

5 DESIGN AND IMPLEMENTATION

A channel of MC-SSL can be categorised according to different criteria. It could be a proxy channel if there is at least one proxy in the channel; otherwise, it is an end-to-end channel. In comparison, the concept of a primary channel and a secondary channel is not as obvious as the previous criterion. As mentioned earlier in Section 3, a primary channel is the first channel in a connection, and it is the basis for secure negotiation and control of secondary channels in the same connection. Besides, a primary channel is actually the channel negotiated by the unchanged SSL protocol. The purpose of secondary channels is to provide multiple cipher suites; therefore, if an application does not need multiple cipher suites but need proxy support of MC-SSL, the implementation of MC-SSL can keep SSL libraries intact (Song, 2004).

In this section, we first discuss the protocol for primary channels in Section 5.1, and then discuss the protocol for secondary channels in Section 5.2. This is also the natural order in an MC-SSL session because secondary channels are "dependents" of primary channels. Due to the space limitation, we omit the description of the simple parts of the protocol, such as the removal of a channel or the modification of channel parameters.

5.1 Primary Channels

Primary channels can be classified into two types of channels: end-to-end and proxy. In an MC-SSL session, there is only one primary end-to-end channel, which is illustrated as channel 1 in Figure 4; on the other hand, there could be a various number of primary proxy channels. Figure 4 only shows one such channel, that is, channel 4.

5.1.1 Primary End-to-end Channel and Primary Single-hop Proxy Channel

We have described the protocol to negotiate and make use of the primary end-to-end channel and primary single-hop proxy channels (Song, 2004). The protocol was referred as MC-SSL proxy protocol because the negotiation of the primary end-to-end channel is simply achieved by the re-use of vanilla SSL. In the next section, we extend a primary single-hop proxy channel to a primary multi-hop proxy channel.

5.1.2 Primary Multi-hop Proxy Channel

A single-hop proxy channel is obviously simpler than a multi-hop one. This is why we always try to combine multiple proxies into one proxy by forming a proxy "cluster" that has a "cluster head" to act as the representative. However, we kept the multi-hop proxy capability of MC-SSL in because multi-hop proxy channel might be necessary due to administrative or security reasons. Also, a multi-hop proxy channel is more general than a single-hop one, and thus makes the model of MC-SSL more complete and versatile.

First, we consider the simplest way to extend the protocol for a single-hop proxy channel (Song, 2004), that is, to iteratively reuse the message interaction between a proxy and a server for the interaction between any two neighbouring proxies shown in Figure 3. There are still some small changes to all the request messages. They are extended to carry information of multiple proxies. In the stage of **C**-**S** handshake, **C** and **S** need to exchange the information about IP addresses (or DNS names), listening TCP ports, and even certificates (or their URLs) of all proxies. Likewise, the request message in the **C**-$\mathbf{P}_1$ handshake is also 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 process continues iteratively until the last proxy ($\mathbf{P}_n$ in Figure 3) connects to **S**.

After the handshake process is finished, every entity in Figure 3 has authenticated its two neighbours if we topologically look at the structure in Figure 3 as a circle. Therefore, **C** can transitionally trust proxies from $\mathbf{P}_2$ to $\mathbf{P}_n$, and **S** has similar transitional trust to proxies from $\mathbf{P}_1$ to $\mathbf{P}_{n-1}$ as well. In the case of a single-hop proxy channel, there is no need for transitional trust because **C** and **S** have authenticated **P** by themselves. However, we believe that transitional trust to proxies is good enough for many applications because proxy channels in MC-SSL are supposed to transport relatively non-sensitive data.

Further, we can enhance this transitional trust model by appending new message interactions to the above handshake process. The enhancement is based on the following two considerations: First, other proxies than $\mathbf{P}_1$ may also require authenticating **C** by themselves, and then authorizing their proxy services accordingly. Our solution is to help **C** distribute its certificate and the verification data if **C** can use a certificate for authentication. Second, **S** or **C** might want to directly authenticate all proxies, in other words, to verify the certificate of every proxy.

In order to satisfy these two requirements of authentication, we have designed a protocol that consists of two stages illustrated in Figure 5(a) and (b). The rationale is to first obtain a random string or number, and then ask **C** and all proxies to "sign" the random string using their own private keys. The "signatures" (verification data) are circulated and used for identity verification.

Figure 5(a) shows the first stage, which purpose is 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 sent by **C** to **P1** contains $\boldsymbol{R}_C$, and finally **C** receives the complete random string from **S**. The purpose of collectively generating a random string is to make sure that no entity can be "deceived" by other entities to accept a random string that is not truly random. This method is actually an extension of SSL, in which only two entities (**C** and **S**) are involved.

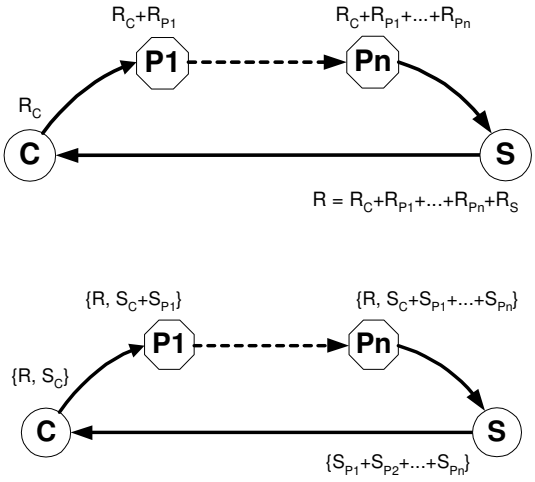

Figure 5. The enhanced authentication: (a) first stage (b) second stage

Figure 5(b) shows the second stage. **C** first sends **P1** a message, which contains the random string generated in the first stage, the certificate (or its URL) of **C**, and the digital signature signed upon $\boldsymbol{R}$ using **C**'s private key. The signature proves that **C** is the owner of the certificate. Each proxy adds a new

signature using its private key corresponding to its own certificate; meanwhile, each proxy can verify **C**'s identity using **C**'s certificate. S_X in figure 6 denotes the signature of entity ***X***. When the message arrives at **S**, it has collected the signatures of all proxies, and therefore **S** can choose to verify them using their certificates. **S** can also forward them to **C** so that **C** can verify them as well.

For the first stage, we add a new field in all the proxy request messages and the **S**-**C** proxy finish message to carry forward the random string, a flag field to indicate if **S** or **C** requests verification of proxies' certificates, and another flag field to indicate if any proxy requests verification of **C**'s certificate. If the flag fields indicate that no verification is required, the second stage will not start. For the second stage, we create a new message called MSG_CP_VERIFICAION to carry all the necessary information. This protocol ends with a MSG_CP_VERI_FINISH message 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 proxies. For example, an application may want to make sure that every chunk of data goes through every proxy in the proxy channel. To achieve data authenticity, we can ask every proxy to "sign" the data chunk using its private key or MAC key. However, that obviously introduces heavy computational burden because **C** or **S** has to deal with every chunk of data. The cost for this kind of protection seems too high for the same reason we mentioned before: a proxy channel is not supposed to transport highly sensitive data unless all the proxies in the channel are highly trustable, and in that case, the transitional trust model without strict authenticity should be sufficient.

5.2 Secondary Channels

Up to this point, the protocols we described cannot provide multiple cipher suites (or multiple channels) inside a point-to-point connection. The only channel we have is the primary channel provided by the unchanged SSL protocol. In this section, we discuss the protocols to negotiate and make use of secondary channels. Every secondary channel can have its own cipher suite and direction different from the corresponding primary channel. In addition, similarly to a primary channel, a secondary channel can also be categorized as a secondary proxy channel or a secondary end-to-end channel.

5.2.1 Negotiation of Secondary Channels

The negotiation process of secondary channels is shown in Figure 6. The negotiation of secondary end-to-end channels (shown in the **A** part) is much simpler than that of secondary proxy channels (shown in **B** part). Two new message formats are added: the secondary channel request message (SEC_CHAN_REQ), and the secondary channel response message (SEC_CHAN_RESP). They are required to be transported through primary channels. In MC-SSL, messages for channel negotiation (or management) must be transported using primary channels so that all channel negotiation is as secure as primary channels, which are provided by SSL. In addition, both the request and the response messages for secondary channels are able to carry requests and responses for multiple secondary channels. The purpose of this design is to reduce repetitive negotiation if an application wants to negotiate more than one secondary channel at some time, especially at the beginning of an application session.

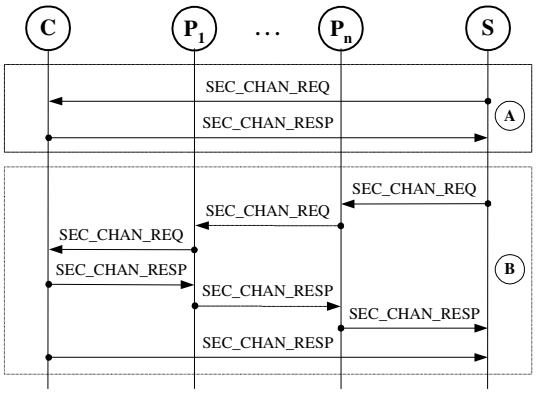

Figure 6. Negotiating secondary channels

In Figure 6, SEC_CHAN_REQ is the request message for multiple secondary channels. For each secondary channel, it carries information as follows: the id of the secondary channel, the id of the collaborative end-to-end channel if the secondary channel is a proxy channel, a list of cipher suites preferred by the message sender, and the data flow direction of the channel. The collaborative end-to-end channel of a proxy channel is the end-to-end channel through which data control messages (APP_DATA_CONTROL_PROXY) are transported when application data is transported through the proxy channel. Please refer to another paper (Song, 2004) for more explanation of the application data protocol. SEC_CHAN_RESP is the response

message for channel requests in SEC_CHAN_REQ message.

For MC-SSL to support multiple cipher suites, we decide to make a small extension to SSL (This will be explained in the next section). However, it is possible that the SSL implementation at **C** and/or **S** does not support the new extension for MC-SSL. In that case, the negotiation of secondary channels will either fail or not start at all.

5.2.2 Extension of SSL for Secondary Channels

SSL is the basis of the design and the implementation of MC-SSL. In order to support multiple channels, we propose to slightly extend current SSL protocol, that is, to add a new field in every SSL packet. The new field is channel id, which indicates the channel that a packet comes from so that the receiver can choose the right cipher suite to decrypt and verify the data fragment encapsulated in the packet. The purpose of channel id is to realize (de)multiplexation inside an SSL connection.

Considering that other new extensions may emerge to require changing the packet format of the SSL record protocol as MC-SSL does, we prefer adding a general extension field so that future extensions or options can be accommodated without change. A similar extension mechanism has been introduced into the client and server hello messages of SSL handshake protocol. RFC 3546 has the details for TLS extensions (Blake-Wilson, 2003). In fact, many Internet protocols such as TCP and IP have fields in their packet formats for options or future extensions.

Because of the introduction of channel id field, several changes regarding SSL protocol and its implementations follow. First, the calculation of the MAC must include the value of channel id field if a SSL packet has a MAC field; therefore, the channel id will not be tampered at will. Second, SSL software must choose the right cipher suite for each incoming packet according to the channel id. The mapping relation between a channel id and a cipher suite is managed by the protocol described in section 5.2.1. Third, some API (Application Programming Interface) functions of SSL 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 different channels.

Obviously, adding a channel id field is a simple and straightforward approach for SSL to support multiple cipher suites. The downside is that the SSL protocol has to be revised and SSL software needs upgrade to a new version. To completely avoid the change of SSL protocol and implementation, we have figured out a possible alternative for SSL libraries such as OpenSSL (OpenSSL, 2004). The basic idea is to "switch" among different channels instead of simultaneously having multiple channels in a SSL connection. The switching of channels is realized by a handshake process implemented in the upper layer codes of MC-SSL instead of the underlying SSL. Basically, one endpoint sends a MC-SSL message to notify the other endpoint what channel it wants to switch to, and then receive the confirmation message from the other endpoint. After that, MC-SSL automatically changes the working cipher suite in the SSL session. However, things are not finished yet. First, it is much harder to switch the working cipher suite for proxy channels, especially multi-hop proxy channels. Second, if we change the working channel to a cleartext channel (null cipher suite without MAC), and then when we want to switch it back to the primary channel, we must make sure that the handshake messages are not tampered by a man-in-the-middle attacker; otherwise, we could be deceived to use a channel that is cryptographically weaker. As a result, we must make one more handshake through the new channel to exchange the MAC data that is calculated upon the previous handshake messages. This approach turns out to require a handshake process that resembles the abbreviated handshake of TLS 1.0 (Dierks, 1999). Such a four-way handshake is inefficient if an application frequently uses different channels, for example, to use different cipher suites for requests and responses. In conclusion, we prefer to add a channel id field in SSL packets than to add a complex handshake process.

5.2.3 Typical Usage of Multiple Cipher Suites

A typical SSL session uses a cipher suite including a 128-bit cipher, and MD5 or SHA-1 hash algorithm for MAC. With MC-SSL we can have an additional channel with only MAC, but without encryption. This channel could be useful for transporting data that does not need confidentiality but require authenticity. Such a channel combination is similar to that of IPSec, in which authentication alone or authentication plus encryption can be separately provided by Authentication Header (AH) and Encapsulating Security Payload (ESP) protocol.

A combination of block cipher and stream cipher is another usage of multiple cipher suites. Streaming media applications such as VoIP telephony 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 one more time or open an additional connection.

Cleartext channels are also available to applications in MC-SSL. Certainly we must be very cautious with a cleartext channel. As a rule of thumb, if a cleartext channel could possibly compromise the security or privacy of a communication session, it should not be negotiated from the beginning of a session. 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 so as 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.

5.3 Prototype

We have developed a prototype to test the ideas and the protocols of MC-SSL. The underlying SSL library we choose is OpenSSL (OpenSSL, 2004). The prototype programs are written in C language and run on Linux. The prototype has helped us improve the protocols, and also demonstrated to us that MC-SSL is feasible and functional.

6 CONCLUSION

In this paper, we present Multiple-Channel SSL, which is a protocol extended from SSL. MC-SSL is able to satisfy diverse security requirements for different applications, especially for emerging mobile or wireless applications.

For the protocol design and implementation of multiple channels over SSL, we extend primary single-hop proxy channels to further support multiple proxies in a proxy channel, and we discuss the protocols for the negotiation and the operation of secondary channels. We also give some thoughts about the typical usage of multiple cipher suites.

Currently, MC-SSL is developed on SSL. One can also apply the multiple channel model of MC-SSL to a security protocol on top of a datagram protocol, such as UDP, so that applications such as VoIP can make use of proxy channels and multiple cipher suites. For instance, 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 without greatly compromising the security. Besides, they can use different cipher suites for user authentication and voice traffic as mentioned in Section 5.2.3. Therefore, we consider it as an interesting research direction to develop a counterpart protocol of MC-SSL for UDP-based applications. For current MC-SSL protocol, we need to further analyse and enhance its security as well as usability.

7 ACKNOWLEDGEMENT

This work was supported 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.

REFERENCES

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

Dierks, T. and Allen, C., 1999. The TLS Protocol Version 1.0. RFC 2246.

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

Kwon, E.K., Cho, Y.G., and Chae, K.J., 2001. Integrated Transport Layer Security: End-to-End Security Model between WTLS and TLS. *Proc. IEEE 15th Int. Conf. on Information Networking*

Kennedy, D. J., 2000. An Architecture for Secure, Client-Driven Deployment of Application-Specific Proxies. Master's Thesis, University of Waterloo.

OASIS Open, 2003. Web Services Security: SOAP Message Security. http://www.oasis-open.org/

OpenSSL, 2004. http://www.openssl.org/

Portmann M. and Seneviratne A., 2001. Selective Security for TLS, *Proc. IEEE 9th Int. Conf. on Networks*, pp. 216-221

Song, Y., Leung, V., and Beznosov, K., 2004. Supporting End-to-end Security Across Proxies with Multiple-Channel SSL. *Proc. IFIP 18th World Computer Congress, submitted paper.*

W3C, 2002. *XML Signature Recommendations*, http://www.w3.org/Signature/

W3C, 2002. *XML Encryption Recommendations*, http://www.w3.org/Encryption/