Design and Implementation of an SSL Component Based on CBD*

Eun-Ae Cho[1], Young-Gab Kim[1], Chang-Joo Moon[2], and Doo-Kwon Baik[1]

[1] Software System Lab. Dept. of Computer Science & Engineering,
Korea University 1, 5-ga, Anam-dong, Seongbuk-gu, Seoul, 136-701, Korea
{eacho,ygkim,baik}@software.korea.ac.kr
[2] Center for Information Security Technologies (CIST),
Korea University 1, 5-ga, Anam-dong, Seongbuk-gu, Seoul, 136-701, Korea
mcjmhj@korea.ac.kr

Abstract. SSL is one of the most popular protocols used on the Internet for secure communications. However SSL protocol has several problems. First, SSL protocol brings considerable burden to the CPU utilization so that performance and speed of the security service is lowered during encryption transaction. Second, SSL protocol can be vulnerable for cryptanalysis due to the fixed algorithm being used. Third, it causes a problem of mutual interaction with other protocols because of the encryption export restriction policy of the U.S. Fourth, it is difficult for developers to learn and use cryptography API for SSL. To solve these problems, in this paper, we propose an SSL component based on CBD. The execution of the SSL component is supported by Confidentiality and Integrity component. It can encrypt data selectively and use various mechanisms such as SEED and HAS-160. Also, it can complement the SSL protocol's problems and, at the same time, take advantage of component. Finally, in the performance analysis, we present a better result than the SSL protocol as the data size is increased.

1 Introduction

In recent years, SSL (secure socket layer) protocol has been mainly used as a security protocol for secure communications over the Internet with OpenSSL by Eric A. Young, JSSE (java secure socket extension) by Sun Microsystems, etc. While SSL is the most common and widely used protocol between web browsers and web servers[1], it has several problems: First, SSL protocol brings considerable burden to the CPU utilization so that performance of the security service in encryption transaction is lowered because it encrypts all data which is transferred between server and client[2][3]. Second, SSL protocol can be vulnerable for cryptanalysis due to the fixed algorithm being used. So, developer cannot use the other mechanisms such as SEED and HAS-160. Third, it causes a problem of mutual interaction with other protocols due to the encryption export restriction policy of the U.S[4]. Finally, it is difficult for developers to learn and use cryptography API (application program interface) for SSL. Hence, we need a new method which is different from the existing one to use SSL protocol more efficiently in design and implementation of applications.

In this paper, we propose an SSL component based on CBD (component based development) in order to solve the problems mentioned above. The component is im-

* This work was supported by the Ministry of Information & Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.

D.-K. Baik (Ed.): AsiaSim 2004, LNAI 3398, pp. 478–486, 2005.

plemented on the application level where it can encrypt and decrypt data. Users can choose various algorithms of encryption. The SSL component provides convenience to the developers who are not accustomed to security concepts. It can also easily provide SSL services when interlocked with other business components because of its component based implementation. The SSL component is reusable and increases the productivity and it decreases the cost[5]. In addition, as component developers need platform-independent methods to support their developments regardless of platform's cryptography APIs, it can support the component platform independently irrespective of the kind of subordinate encryption APIs. As a result, the component makes it easy to interact and interlink between protocols.

In this paper, we propose the requirements for SSL component and design it based on CBD. We have implemented internally SSL handshake protocol and SSL record protocol in order to perform the same functions of the existing SSL implementations. Further, we designed the main security component – Integrity and Confidentiality service component – that supports the part of SSL component. Here, we add standard algorithms used in Korea for cryptography such as SEED[6] and HAS-160[7] and developers using the component can select the algorithm type and encoding/decoding type. In other words, we provide variety cryptography mechanisms of SSL and implementations to encrypt/decrypt data selectively, thus rendering data processing to be more efficient. We used the Rational Rose 2000 as a design tool for the component and sequence diagram[8], and implemented the SSL, Confidentiality and Integrity component with EJB (enterprise java beans)[9]. Lastly, we tested the implementations of each component with the appropriate scenario for SSL component and J2EE (java 2 platform, enterprise edition)[10] server according to our proposed design. We also tested the efficiency against the standard SSL protocol.

The remainder of this paper is organized as follows. Chapter 2 explains the analysis of requirements for SSL component and presents the proposed design of SSL component, which solves the aforementioned problems. Chapter 3 presents the implementation of SSL and the main security component according to the design of Chapter 2, and then provides the test through the proper scenario. Further, performance evaluation results compared with SSL protocol are presented in Chapter 3. Finally, Chapter 4 concludes the paper with a brief discussion on the future work.

2 Design of SSL Component

In this chapter, we propose the requirements for the SSL component derived from the problems of the SSL protocol, and design the SSL component and the main security component (Confidentiality and Integrity service component). Then, we define the main methods of the SSL component, and describe the whole movement and flow of the messages between Confidentiality component, Integrity component, and SSL component using a sequence diagram.

2.1 Requirements for SSL Component

We classified requirements for SSL component into two parts: first are requirements for solving the SSL protocol problems, and second are requirements for the implementation of SSL component based on CBD.

1) Requirements for Solving the SSL Protocol Problems

Our purpose is to provide both convenience and compatibility and at the same time, the performance should be improved without affecting the security issue.

- By encrypting data selectively, the amount of data processed by CPU should be reduced without affecting the security problem.
- The SSL component should be standardized in order not to depend on a specific mechanism, but it should be able to provide proper security mechanism according to the encryption level.
- The security requirements such as confidentiality and integrity for SSL component platform should be supported apart from the subordinate cryptography API in a system.
- Programming which is related to security should be required at a minimum in the code of application component, which exist or will be developed.

2) Requirements for Implementation of SSL Component Based on CBD

To implement the SSL component, which supports confidentiality and integrity in J2EE platform environment, the following requirements are needed:

- It should be ensured that the reusability of SSL component using standardized security component interface(IConfidentiality, IIntegrity) in business components.
- An application component developer should be able to detect the subordinate security component within the component code and set the desired security component by the algorithm name.
- When message protocol (e.g. handshake, record protocol, etc.) is implemented, it has to support the order of the messages which is defined in SSL standard.
- In this paper, SSL component focused on implementation of confidentiality and integrity function by using anonymous Diffie-Hellman algorithm for key exchange in the server's handshake protocol.
- It should create the MAC and perform encryption/decryption using the confidentiality/integrity component implemented with standardized security component interface in the record protocol.
- It should maintain and/or manage the session state parameter and the connection state parameter between client and server environment, where it has an end-to-end connection in SSL component. Thus, SSL component should be implemented with EJB code in the form of the session bean that its inner states consist of a session state and a connection state.
- It has to support Korea standard cryptograph algorithm such as SEED and HAS-160.

2.2 Design of SSL Component

To satisfy the function mentioned in the requirements above, we designed and implemented the component based on EJB. The whole SSL component based on CBD is composed of main security components such as confidentiality and integrity component as shown in Fig. 1.

Table 1 shows the major interface of SSL component based on the requirements. According to SSL component interface, we designed EJB Component, which is called SSLComponent to perform the same function as SSL protocol.

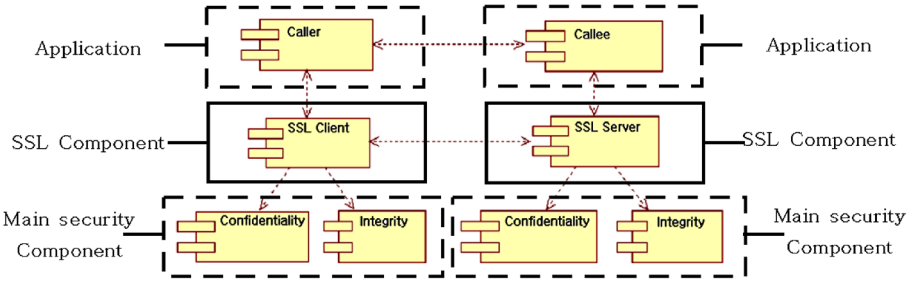

Fig. 1. Component Diagram of SSL Component

Table 1. Description of the main method

Component Name	Main Method	
	Name	**Description**
SSL Component	startHandshake()	To exchange the session key and algorithm, server and client start handshake protocol.
	clientHello()	To start handshake protocol, client calls server.
	Request-KeyExchange()	To exchange the key, client transfers its public key to server.
	getPubKey()	To create the public key for the other part.
	encryptSSLMessage()	After handshake, both server and client encrypt the real data to SSL message using the set key and algorithm.
	decryptSSLMessage()	After handshake, both server and client decrypt SSL message to the real data using the set key and algorithm.
	finishHandshake()	Both server and client send the current cipher spec, and set up the security connection.
	decisionAlgorithm()	When helloMessage is sent, server decides the algorithm of confidentiality and integrity component for current session.
	createMasterSecret()	Both server and client create the master key to share the algorithm and key for SSL.
	preMaster()	Both server and client create the pre-master key using their own private key and the other part's public key which is came from requestKeyExchange().
	setAlgorithm()	To set a proper algorithm.
Integrity Component and Confidentiality Component	getAvailable-Algorithms()	To get all available algorithm presently.
	getAlgorithm()	To get the algorithm that is currently set.
	setNameType()	To set the type of the algorithm name either OID(object identifiers) or Name.
	getNameType()	To get the type of the current algorithm name.
	initialize()	To receive the key value and initiation value by the parameter and then performs initialization.
	update()	To receive the message made for byte array and then encrypts/decrypts the data as much as inner buffer allows.
	Finalize()	To pass the result for all message encryption including the remainder of message in fixed buffer.
	getLength()	To get the length of the encrypted message.
	setEncodingType()	To set the encoding type either Raw or Base64.
	getEncodingType()	To get the current encoding type.

SSL protocol provides the confidentiality, integrity and authentication function (non-reputation is added at the user application level) in general[11][12]. In this paper, we first implement a component, which provides the confidentiality and integrity service. Then, we propose an SSLComponent. Key exchange uses anonymous Diffie-Hellman algorithm for exchanging a key, therefore we made the concept of authentication simple.

Confidentiality is an information security service, which ensures that other people cannot find out the content of message online and/or offline, and integrity is an information security service that prevents other people illegally creating, changing or deleting the content of information transferred via the network.

Fig. 2 shows the actions between SSLComponent and confidentiality component in the sequence diagram. It shows diagram that a message is encrypted or decrypted for enabling confidentiality. In Fig. 2, on the left is the encryption process and on the right is the decryption process for message. The kind and order of the message received and sent is the same in the confidentiality component. First, SSLComponent creates the home interface of confidentiality component, and checks available algorithms, and then exchange current algorithm and finally selects the algorithm to be used. Type of algorithm is selected can be chosen either as by the algorithm name or algorithm OID chosen by the developer. After the algorithm selection, we can also choose the encoding type either as Raw or Base64. After these processes, SSLComponent sends the encrypted message through the process such as *initialize*, *update* and *finalize* process.

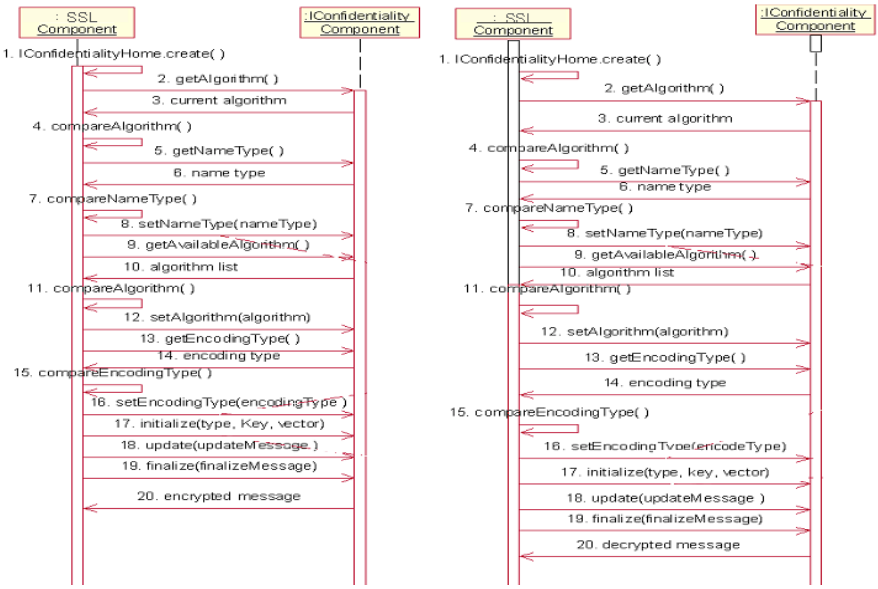

Fig. 2. Sequence diagram of confidentiality component

Fig. 3 presents the message process between SSLComponent and Integrity component with the sequence diagram. It acts on the same message and order as confidentiality component.

We designed the EJB component called SSLComponent that behaves according to the SSL protocol and according to the definition of SSLComponent interface. We implemented the SSL record protocol and SSL handshake protocol as a part of the SSLComponent, which provides the confidentiality and integrity service.

Fig. 4 shows the whole message flow according to the role of each SSLComponent by the sequence diagram. We can classify the roles into SSLServer and SSLClient.

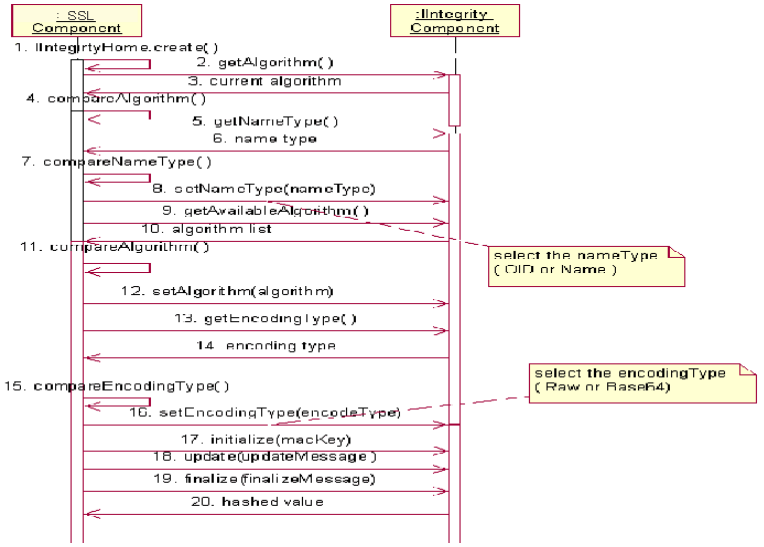

Fig. 3. Sequence diagram of integrity component

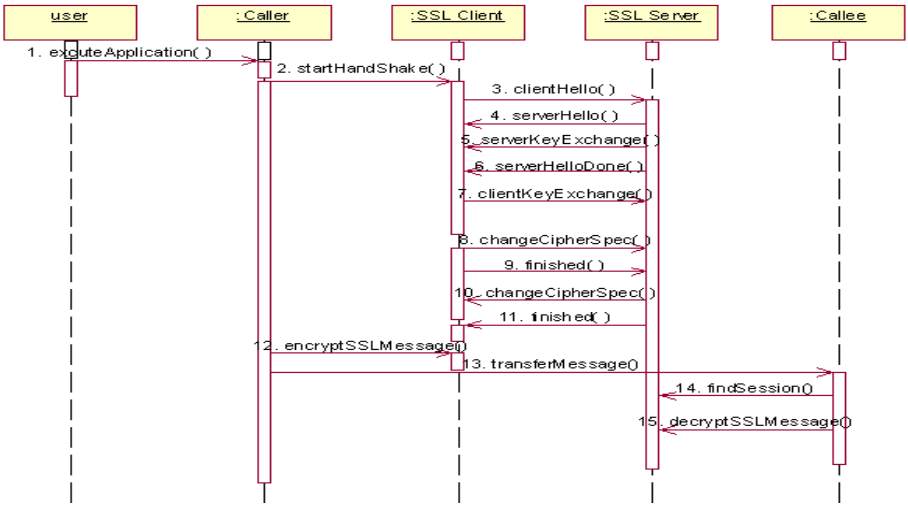

Fig. 4. Sequence Diagram of SSLComponent

Through the steps 2 to 11, SSLClient and SSLServer initialize the logical connection and execute the handshake protocol. The steps 5 to 7 show the asymmetric key exchange between SSLClient and the SSLServer. We omitted the key exchange method because anonymous Diffie-Hellman was used. The steps 9 to 11 are the process to complete the security connection. The steps 12 to 15 are SSL record protocol.

3 Implementation and Performance Evaluation of SSLComponent

In this chapter, we propose the implementation based on CBD to overcome the limitation of the problems of the existing SSL protocol. The main differences between proposed SSLComponent and existing SSL protocol are shown in Table 2.

It is impossible to reuse existing SSL protocol within other software, since once the SSL channel is set, all the data transported between the server and the client, is encrypted. Thus, we can't provide the selective encryption. However SSLComponent can be reused and provide the SSL service selectively for only certain messages. Thus it is possible to customize the message transmission device according to the development plan. In addition, it is precarious to ensure safety since existing SSL protocol doesn't have the flexible mechanism. On the other hand, SSLComponent does not rely on a specific mechanism. Various security algorithms and mechanisms can be selected and used including domestically standard cryptography algorithms used in Korea according to developers' preference. Furthermore, it is possible to extend the SSL protocol function in the form of the selective SSL service.

Table 2. SSLComponent Compared with SSL protocol

	SSLComponent	Existed SSL protocol
Reusability	Can be reused.	Cannot be reused.
Flexibility	Can select the algorithm and key according to developer's intention with confidence.	Use the fixed algorithm and fixed key length.
Variety	Can apply and select the various security mechanisms.(Also it can use standard Korean crypto algorithms-SEED, HAS-160.)	Cannot select the mechanism. (It cannot use standard Korean algorithms.)
Extensibility	As an option, it can extend the SSL service.	Cannot extend connection previously set.
Generality	Developer can develop a system without the prior knowledge related to the cryptography API.	Developer needs to be familiar with the cryptography API.
Efficiency	Can reduce the CPU overhead and time.	Reduce efficiency due to processing of the whole data.

There is a problem that the SSL protocol could be implemented only by the developers who are fully aware of cryptography API. And sometimes the protocol even lowered the performance because it processes and encrypts the whole data during the SSL process. But SSLComponent can be developed by the developers who are not even aware of API because the component uses a common and standard form. In addition, using the proposed component, CPU utilization overhead can be reduced as

well as the time overhead, because the developer can decide the API and the algorithm selectively.

SSLComponent were built on the test server. Test server has the following configuration: Xeon 2.2GHz Processor, 512KB L2 ECC Cache, 512MB ECC SDRAM Memory, 36.4GB Pluggable Ultra320 SCSI 10k Universal Disk and Compaq 7760 10/100/1000 TX PCI NIC LAN Card.

We have chosen a scenario of online shopping (buying goods) to compare the proposed SSLComponent efficiency with existing SSL protocol. The criterion for comparison is the process time, which is needed for safe transmission of the data. For performance evaluation, we tested both SSLComponent and existing SSL protocol on the same server hardware.

Although same server hardware was used, computing environments were different for the two protocols being compared. In other words, SSL protocol is mainly used as https in web server like Apache, while SSLComponent is executed in an application server like J2EE based on Java. The Java environment may have the disadvantage in the time factor due to virtual machine it is running on. Thus we had to consider those differences for the clear and correct performance evaluation between both. We have tested SSL protocol in Apache server environment and SSLComponent in J2EE server environment for performance measurement. Fig. 5 shows the comparison result between the pure Apache Tomcat 5.0 and the pure J2SDKEE1.3.1 server. For equivalent application of execution environment, we take their difference into consideration and perform our tests.

The difference value is calculated by subtracting Apache web server process value from J2SDKEE1.3.1 server. The data is represented by function **(1)**

$$y=159.62\ln(x)-766.3 \tag{1}$$

y is the value of subtracting Apache web server from J2SDKEE1.3.1 server and x is the value of data size.

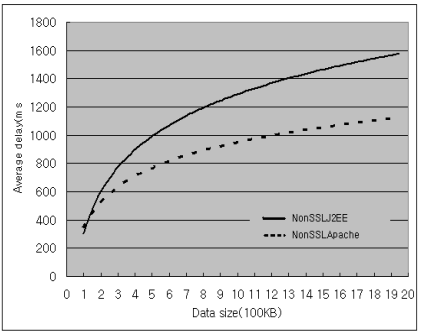

Fig. 5. J2EE Compared with Apache

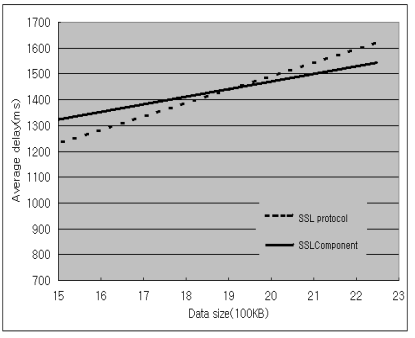

Fig. 6. SSLComponent Compared with Https

Fig. 6 shows the comparison between the execution result of SSLComponent in J2EE server and the execution result of https protocol using SSL protocol in Apache server. As shown in the graph, server's process time is proportioned to the data volume. When data volume is small, data processing time is a little bit longer than component performance time, but the more data volume is increased, the more the time gap is narrowed. When data volume is very big, the processing time of SSLCompo-

nent is shorter than https. On the other hand, the processing time for a small data volume is not significantly improved. However, SSLComponent does not have any worse performance efficiency than the SSL protocol, and the processing time for large data is better than SSL protocol.

4 Conclusion and Future Works

In this paper, we proposed and implemented the model to provide the SSL service through the system based on CBD. SSLComponent model based on CBD extends the existing SSL protocol through the use of partial message encryption and Korean domestic standard cryptography algorithms which wasn't provided in the existing SSL protocol. We also overcame the limitation of SSL protocol(e.g. it cannot be reused, has atomic property, cannot use Korean domestic standard algorithm, cannot extend, and etc.) through the software development which has the component concept. The SSLComponent can be reused, encrypted selectively, applied for Korean domestic standard algorithm and it can be extended.

As showed in the chapter 3, when the size of data becomes small we can reduce the need for CPU resources by using the remote connection of a component. On the other hand, when the data volume size is increased, proposed SSLComponent becomes more efficient, since it can encrypt the selected data based on CBD. In this paper, we used anonymous Diffie-Hellman algorithm, which is quite simple. As for future work, we need to propose and design the complex authentication part more clearly. Further, we need to present and implement the standard for the component that would provide other security services such as non- reputation and availability.

References

1. A. Freier, P Karlton, and P. Kocher: The SSL Protocol Version 3.0, Internet Draft (1996)
2. Xiaodong Lin, Johnny W. Wong, Weidong Kou: Performance Analysis of Secure Web Server Based on SSL. Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 1975/2000, Information Security: Third International Workshop, ISW 2000, Wollongong, Australia, December 2000. Proceedings (2003) 249-261
3. K. Kant, R. Iyer and P. Mohapatra: Architectural Impact of Secure Socket Layer on Internet Servers. Proc. IEEE 2000 International Conference on Computer Design (2000) 7-14
4. Kyoung-gu, Lee: TLS Standard Trend. KISA, The news of Information Security, vol. 19 (1999)
5. Chris Frye: Understanding Components. Andersen Consulting Knowledge Xchange (1998)
6. KISA: SEED Algorithm Specification. Korea Information Security Agency (1999)
7. TTA Standard: Hash Function Standard-Part 2: Hash Function Algorithm Standard(HAS-160). Telecommunications Technology Association (2000)
8. Booch, G., Rumbaugh, J., and Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley Longman (1999)
9. Enterprise Java Beans Specification Version 2.0 Final Release. Sun Microsystems Inc (2001)
10. Sun, Java 2 Platform Enterprise Edition Specification, Version 1.4, Sun Microsystems Inc (2004)
11. William Stallings: Cryptography and Network Security. Principles and Practice, 3rd edn, Prentice Hall (2002)
12. R. W. Badlwin et C. V. Chang: Locking the e-safe. IEEE Spectrum (1997)