Information Security Component Framework and Interfaces for Implementation of SSL

Jong-Whoi Shin
jshin@kisa.or.kr
Korea Information Security Agency, Seoul, Korea

Summary
Various security APIs such as IETF *GCS-API* and *GSS-API*, RSA *Cryptoki*, Microsoft *CryptoAPI*, and Intel *CDSA* are being used in a variety of application areas requiring the information security function. However, these proprietary standards are not compatible, and the developer must use those APIs selectively depending on the application environment or the programming language. To resolve this problem, we propose the information security component framework, while SSL (Secure Sockets Layer) using the confidentiality and integrity component interfaces has been implemented to verify validity of the framework. The implemented SSL uses the lower-level SSL component when establishing the RMI (Remote Method Invocation) communication between components, as if the security algorithm had been implemented by adding one more layer on the TCP/IP.

Key words:
Component Based Design, Application Programming Interface, Secure Socket Layer, Remote Method Invocation.

1. Introduction

A Various APIs such as IETF *GCS-API* and *GSS-API*, RSA *Cryptoki*, Microsoft *CryptoAPI*, and Intel *CDSA* are being used in a variety of application areas requiring the information security function [1-4]. Each proprietary standard was developed to fit the application area of each vendor, as integrating security APIs that were designed for different purposes and usages was out of the question. In other words, various security APIs were all developed separately, have experienced some difficulty in adopting the application in the distributed environment due to a lack of compatibility. To resolve this problem, we introduced a component design technique because: 1) the component ensures the enhanced timeliness and productivity of development by preparing the software for the various requirements, and also 2) it encourages standardization between various products.

This paper proposes the information security component framework and interfaces in order to bring about the compatibility of the information security functions, and efforts are being made to standardize the interface of the information security components. To verify the validity and stability of the framework development, the SSL (Secure Sockets Layer) was implemented using SSL v3.0 specification in the J2EE environment. Finally, we confirmed the validity of our proposal through demonstrating the book shopping service. This paper is organized as follows: in section 2, the information security service component interface has been described, while in section 3 the design and implementation of the SSL component have been outlined. The conclusions drawn are detailed in section 4.

Manuscript received October 5, 2007.
Manuscript revised October 20, 2007.

2. Information Security Service Component Interface

2.1 Information Security Component

The information security component is an independent software component with more than one function to support the information security service at each IT application area. It is designed to provide the core information security functions of the current security API such as confidentiality, integrity, authentication, access control, and non-repudiation [5, 6].

- *Confidentiality service component*

Confidentiality is necessary to conceal important information that is saved or transmitted in online and offline environments from an unauthorized or unidentified party. This component provides encryption and decryption functions based on the conventional encryption algorithm and the public key encryption algorithm.

- *Integrity service component*

Integrity is required to protect information content transmitted via the network from being illegally created, modified, or deleted. This component provides a hash algorithm and a MAC (Message Authentication Code) generation function.

- *Authentication service component*

Authentication is required to clearly identify an entity through information exchange. This component provides a digital signature algorithm and an integrity service function.

Table 1 : Descriptions of Component Layers

Layers	Sub-layers	Descriptions
Domain Layer	Domain Specific	Components for application areas (Manufacturing, Financing etc.)
	Domain Common	Components used for the specific domains commonly
Common Layer	Business Management	Components used for e-biz, XML/EDI domains etc. with business logic
	Business Common	Components used for several domains commonly without business logic
Platform Layer	None	Component Architecture such as EJB, COM+, CCM

● *Access control service component*
This component provides the authentication service function and prevents unauthorized usage of the resources by controlling access based on access rights.

● *Non-repudiation service component*
This component is used to prevent the repudiation of a transmission and its content between the sender and receiver. It provides the authentication service function.

The component should be used with reliability, by ensuring the stability of the information security component itself, when using the information security function based on the component. While the component has scalability and universality by nature, the component itself may contain security vulnerability, which may also be caused at the component composition stage. To secure component security, and to use it as the information security component, the vulnerability possibility should be removed. Currently, many studies are being carried out on this subject. The framework proposed by this study stresses the need to minimize this problem when creating the component, by using the interface. To this end, the pre-conditions and post-conditions are clearly presented for the I/O stage of each interface.

2.2 The Information Security Component Framework

Since general users are flooded with many security APIs, there is some confusion about the layer location of the information security component and the location of the developed component for the interface. Fig. 1 shows the information security component framework that was designed with a 3 tier layer (basic distributed environment structure) and the hierarchical structure of the component area. To provide the information security service to every component area in this hierarchical structure, the information security component is included in the general

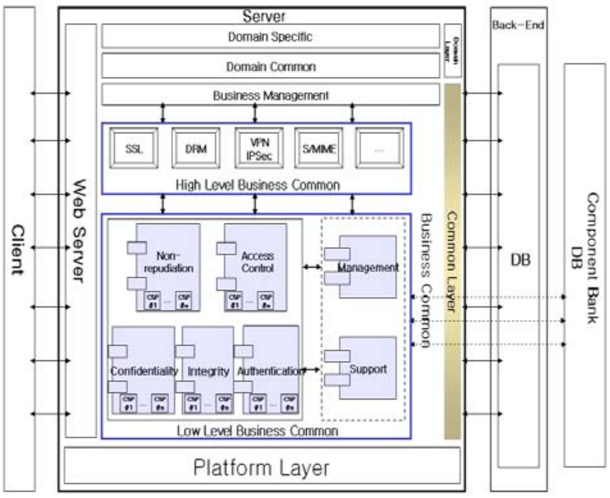


Fig. 1 Information Security Component Framework

business layer. Various component-based information security products that belong to the upper layer have been implemented with the basic information security components provided by the information security component framework, such as confidentiality, integrity, authentication, non-repudiation, and the access control components. Information security components located at the general business layer in the information security component framework provide the information security service function for the components at another layer.

2.3 The Confidentiality and Integrity Service Component Interfaces

The confidentiality and integrity service component interfaces, which are located at the low level business common layer, provide a service that encrypts/decrypts or digest the message to provide the confidentiality and integrity functions. Design is based on the IDL (Interface Definition Language) interface to secure universality on the different platform. Several methods can be used to define the interface. IDL is not needed if *Visibroker Caffein* is used, or the interface can be defined without knowing the IDL if Java RMI or RMI-IIOP is used. However, in order to secure availability, which is one of the requirements with which the information security component should be equipped, the easy to learn standard interface definition language as defined by the OMG (Object Management Group) [7]. The IDL was used to specify *IConfidentiality* and *IIntegrity*– the confidentiality and integrity service components. These interface algorithms are represented in Table 2 and 3.

In these algorithms, SetAlgorithm operation sets the name or the identifier of the encryption/description or digest

Table 2 : Algorithm of the Confidentiality Service Component Interface (IConfidentiality)

```
exception UnSupport {string why};
    exception IllegalState {string why};
    exception InvalidState {string why};
    exception InternalState {string why};
void setAlgorithm(in string cipherAlgorithm)
out string[] getAvailableAlgorithms()
out string getAlgorithm()
void initialize(in string opMode, in byte[] key, in
byte[] iv)
out byte[] update(in byte[] message)
out byte[] finalize(in byte[] message)
out int getLength(in int inputLength)
void setEncodingType(in string codingType)
out string getEncodingType()
void setNameType(in string IDType)
out string getNameType()
```

Table 3 : Algorithm of the Integrity Service Component Interface (IIntegrity)

```
exception UnSupport {string why};
    exception IllegalState {string why};
    exception InvalidState {string why};
    exception InternalState {string why};
void setAlgorithm(in string digestAlgorithm)
out string[] getAvailableAlgorithms()
out string getAlgorithm()
void initialize(in byte[] macKey)
void update(in byte[] message)
out byte[] finalize(in byte[] message)
out int getLength()
void setEncodingType(in string codingType)
out string getEncodingType()
void setNameType(in string IDType)
out string getNameType()
```

algorithm to be used as the confidentiality and integrity functions, with the pre-condition that the value of the cipherAlgorithm and digestAlgorithm should be OID, general name or NULL, and that the components should be initialized before calling operation. The post-condition is that proper exception handling should be performed – IllegalState if the component is not initialized, UnSupport if the supporting algorithm is not available, or InternalState if the internal problem occurs inside the component. Fig. 2 and 3 show the sequence diagram of the confidentiality and integrity components.

3. Designing and Implementing the SSL Component

3.1 SSL Provided by Current Components

As more components are used, the demand for security concerning component usage increases, and the platform development companies begin to support the security solutions at the server level to accommodate those requirements. Authentication, authorization, and usage of the SSL (Secure Sockets Layer) constitute the security issues in the component platform. The SSL component is located at the high level business common layer according to our definition. The following methods can be used to set the security in the component platform [8].

- *Declarative method*

When the security requirement is set by the deploy tool, the container handles the security requirements properly in the manner of a transaction handling, without affecting the code.

- *Programming method*

Setting the security requirements by programming requires application of the security requirements at the code level. Each time the code is modified, compilation and execution should be performed again. A fine-grained security setting is possible. Selecting one of two methodologies is a matter of maintaining the balance of convenience and control. SSL is a security communication protocol developed by Netscape to provide confidentiality and integrity through mutual authentication and encryption in the TCP/IP.

Generally, SSL is included in the browser and web server, and used for various web communication protocols with the HTTPS format. If the new SSL component is implemented, fine-grained security can be set: discovery and interface in the business component are easy to use, and various encryption mechanisms including the local encryption algorithm can be set according to the security policy of the platform.

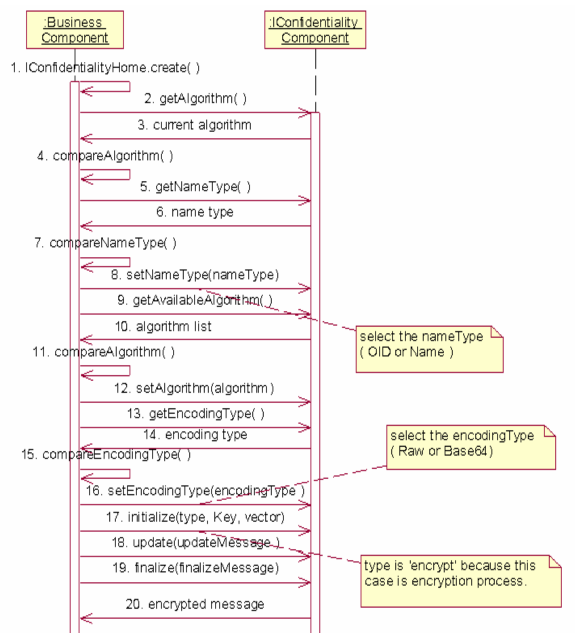

:Business Component
:IConfidentiality Component
1. IConfidentialityHome.create( )
2. getAlgorithm( )
3. current algorithm
4. compareAlgorithm( )
5. getNameType( )
6. name type
7. compareNameType( )
8. setNameType(nameType)
9. getAvailableAlgorithm( )
10. algorithm list
11. compareAlgorithm( )
12. setAlgorithm(algorithm)
13. getEncodingType( )
14. encoding type
15. compareEncodingType( )
16. setEncodingType(encodingType)
17. initialize(type, Key, vector)
18. update(updateMessage)
19. finalize(finalizeMessage)
20. encrypted message
select the nameType (OID or Name)
select the encodingType (Raw or Base64)
type is 'encrypt' because this case is encryption process.

Fig. 2 Sequence Diagram of the Confidentiality Component

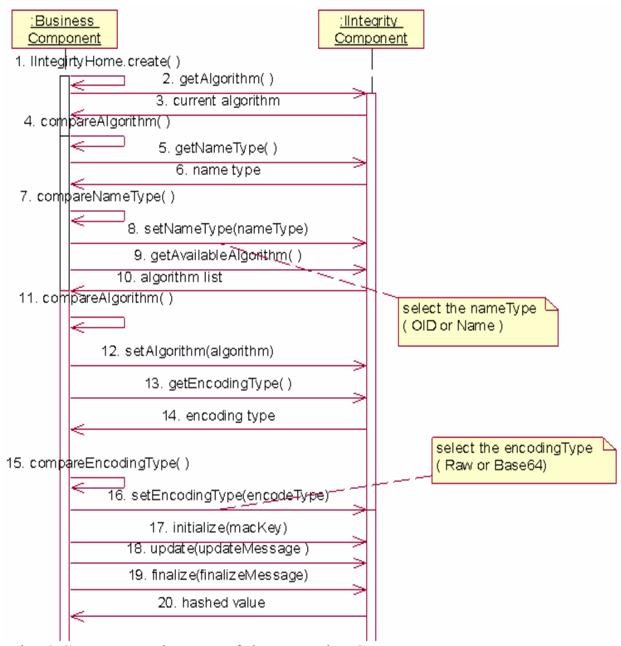


Fig. 3 Sequence Diagram of the Integrity Component

3.2 SSL based on the Confidentiality and Integrity Service Components

In this structure (Fig. 4), the SSL components at the lower level are used when establishing the RMI communication between components, as the security algorithm is implemented in SSL by adding one more step to the TCP/IP. When the Caller component requests the security connection to the Callee component and uses the SSL, the Callee requests the creation of the SSLServer component, and the Caller creates the SSLClient component, and exchanges the encryption algorithm and session key that will be used by the client and server via the SSL handshake protocol. When the handshaking process is complete, RMI parameters and the actual message equal to the result value are encrypted/decrypted using the encryption algorithm that was exchanged, and the security of the exchanged message is provided by creating/verifying the MAC.

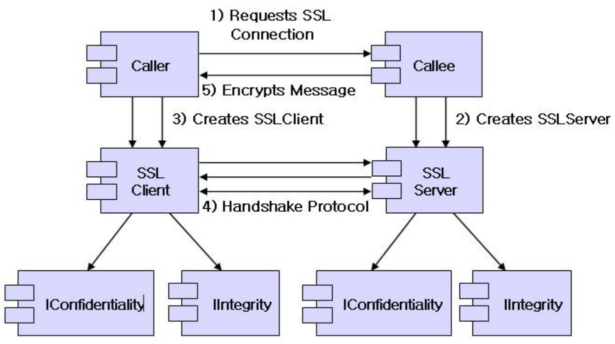


Fig. 4 Transmission of the RMI-based SSL Message

The execution methods are as follows :
1) The caller requests SSL connection (secure connection) to callee.
2) Callee creates the SSLServer component object.
3) Caller creates the SSLClient component.
4) Execute the handshaking protocol between SSLClient and SSLServer.
5) Encrypts the RMI transmission message, using the handshaking result.

When execution of the SSL handshaking protocol is complete, the messages are encrypted or transferred as the parameter by calculating the MAC using *IConfidentiality* and *IIntegrity* based on the SSLClient component algorithm, which was decided when calling by the methods of the Callee component and Caller component.

Likewise, Caller verifies the MAC using the defined algorithm to ensure integrity, and decrypts the received message using the defined encryption algorithm. The same method is used when sending the processing result of the callee to the caller. This implementation method is not limited by the specific component platform environment, since the lower TCP/IP socket is not directly used.

This approach has two advantages: 1) it is possible to implement the security communication only using RMI of EJB, and 2) it does not limited to EJB environment due to using TCP/IP socket indirectly. Fig. 5 shows the sequence diagram of SSL and Fig. 6 shows the deployment of *IConfidentiality* and *IIntegrity* Components.

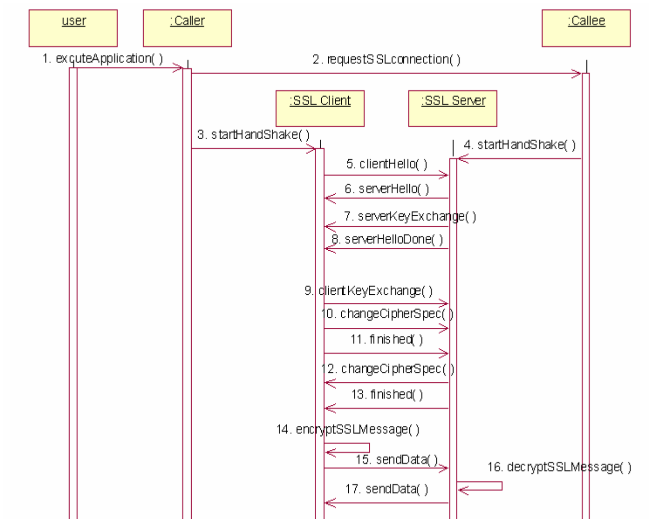


Fig. 5. Sequence Diagram of SSL

In the result of testing for correctness, the implemented SSL using our component interfaces carried out correctly the confidentiality and integrity functions without errors. Fig. 7 shows the result of financial transaction using SSL component in book shopping service. Consequently, the number of orders and payment was accurately fulfilled. The implementation environments are shown in the Table 4.

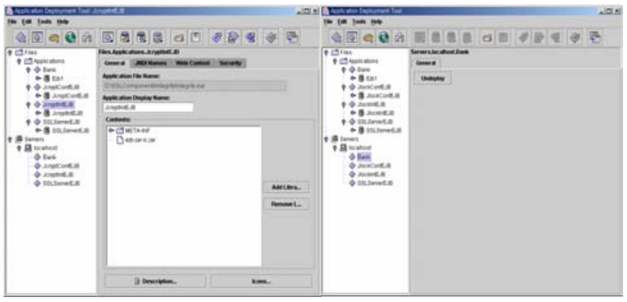
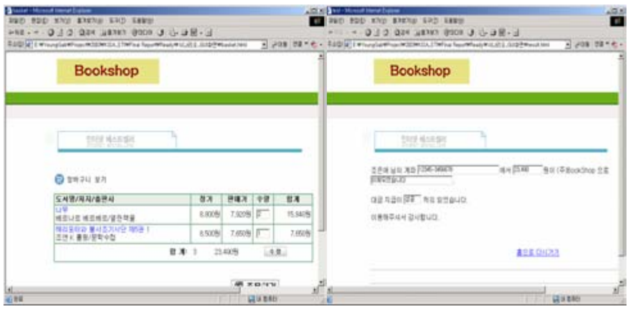


Fig. 6. Deployment of *IConfidentiality* and *IIntegrity* Components



a) Ordering Process b) Payment Process

Fig. 7. Book Shopping Service using SSL component

Table 4. Implementation Environments

Types	Tools
Test Server	Window 2000
Programming Language	J2SDK 1.4
Web and EJB server	J2EE
Encryption Library	J/LOCK (Proprietary)
DBMS	Oracle 9i

4. Conclusion

This paper proposed the information security component framework for compatibility of security APIs by introducing the component design technique. To verify the validity of the proposed framework, the method for implementing SSL v3.0 using the EJB component in the J2EE environment has been described. By introducing a component design technique that is more advanced than the current security APIs, the quality of the software requiring the security service should be ensured. Finally, we expect that the proposed framework is to enhance system development productivity and compatibility.

References

[1] S J. kabat, M. Upadhyay, "Generic Security Service API Version 2 : Java Bindings", IETF RFC2853, June 2000.
[2] RSA Lab., "PKCS #11: Cryptographic Token Interface Standard", RSA, August 28, 2002
[3] V. Smyslov, "Simple Cryptographic Program Interface (Crypto API).", IETF RFC 2628, June 1999.
[4] TOG, "Common Security: CDSA and CSSM, Version 2", Open Group, May 2000.
[5] ISO 7498-2, "Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture", 1989.
[6] S. A. Hissam, D. Carney, D. Plakosh, DoD Security Needs and COTS-Based Systems, Carnegie Mellon SEI, Sept. 1998.
[7] OMG Laboratories, "OMG IDL Syntax and Semantics", OMG, July 2002.
[8] SUN, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Security.html

Jong-Whoi Shin received M.S. and Ph.D degrees in Computer Science and Technology from Korea University, South Korea, in 2001 and 2007, respectively. He is working for the Korea Information Security Agency as a principal researcher. His research interests include security for mobile ad hoc wireless networks, ubiquitous sensor networks, security APIs and intrusion tolerant systems.