Model-based Design of Reusable Secure Connectors

Michael Shin
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
michael.shin@ttu.edu

Hassan Gomaa
Department of Computer Science
George Mason University
Fairfax, Virginia, USA
hgomaa@gmu.edu

Don Pathirage
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
don.pathirage@ttu.edu

***Abstract*—This paper describes the integration of security and communication patterns in reusable secure connectors that are incorporated in the model-based design of secure distributed component-based software architectures. The secure connectors are designed separately from application components by reusing the appropriate communication pattern between components as well as the security patterns required by these components. Each secure connector is designed as a composite component that encapsulates both security pattern and communication pattern components. Integration of security patterns and communication patterns within a secure connector is enabled by a security coordinator. The main advantage is that secure connectors can be reused in different secure applications.**

***Index Terms*—Reusable secure connector, secure software architecture, component-based software architecture, secure software design, message communication patterns, security patterns, dynamic modeling, model-based design, UML.**

I. Introduction

A component-based software architecture (CBSA) [12] for distributed applications is designed by means of components, which encapsulate the functionality of the application, and connectors, which encapsulate the details of inter-component communication, such as asynchronous communication or synchronous communication with reply. UML 2 provides a notation for modeling components, ports, provided and required interfaces in CBSAs [16]. A CBSA can be developed using a model-based design method as described in [5].

Although connectors are typically used in CBSAs to encapsulate communication mechanisms between components, this paper describes how security concerns can also be encapsulated in software connectors, which are referred to as secure connectors, separately from application components that contain application logic. To effectively integrate security concerns with communication concerns, it is necessary to design secure connectors that are both modular and reusable.

Each secure connector is designed as a composite component using CBSA concepts by reusing security pattern components and communication pattern components, which are designed separately from each other. Each security pattern component encapsulates a security pattern, such as symmetric encryption or digital signature. Each communication pattern component encapsulates the communication pattern between application components, such as synchronous or asynchronous message communication. A secure connector is then constructed by composing security pattern components and communication pattern components. Integration of security patterns and communication patterns within a secure connector is provided by a security coordinator. Once a secure connector is constructed, it can then be reused in different secure applications.

This paper describes the integration of security patterns and communication patterns within reusable secure connectors that are used in distributed secure CBSAs using the UML notation. Each reusable secure connector in previous papers by the authors [9, 10, 11] was described with one or more security services accessed by a security coordinator, the template of which was not addressed. This paper extends the reusable secure connectors in [9, 10, 11] by describing security patterns for security services and a pseudocode template for the high-level security coordinator that is customized for each secure connector based on the security pattern(s) selected. A security pattern addresses a specific security technique that realizes a security service. Reusable secure connectors make complex software applications more maintainable by separating security concerns from application concerns in the software architectures. Reusable secure connectors described in this paper are applied to the software architectures for electronic commerce applications.

In this paper, section II describes related work, section III describes reusable secure connectors, section IV describes secure asynchronous message communication connector, followed by validation of reusable secure connectors in section V. Section VI concludes the paper.

II. Related Work

Related work focuses on approaches to designing software architectures for secure applications and patterns for distributed communication. The authors in [13] proposed SecureUML, which is a new modeling language based on UML for the model-driven development of secure systems. Work has also been proposed to provide an extension of UML called UMLsec [14] that helps with the expression of security-relevant information within design diagrams. In Model-driven security [3], a system is modeled with its security requirements and security infrastructures are generated using the models.

Using connectors as the central construct, a distributed CBSA in [5, 15] is composed of a set of components and a set of connectors that can be used to connect the components. In

[6], a connector centric approach is used to model, capture, and enforce security. The security characteristics of a CBSA are described and enforced using software connectors. Methods in [1] propose SecArch to evaluate architectures with significant security concerns.

Security patterns in [4, 7] address the broad range of security issues that should be taken into account in the stages of software development lifecycle. The authors describe the problem, context, solution, and implementation of security patterns in a structured way with a template so that the presentations are consistent. The security patterns can help developers to construct secure systems, even though the developers may not have security expertise.

In recent work by the authors [9] described secure asynchronous and synchronous connectors for modeling the software architectures for distributed applications and the design of reusable secure connectors that are structured into reusable security components and communication components. The authors in [11] address the design of secure connectors in terms of maintainability and evolution, which are used in the design of secure software architectures. One of the authors has also investigated designing dynamically adaptable and recoverable connectors [17, 18].

III. Reusable Secure Connectors

In a CBSA [12] for concurrent and distributed applications, components address the functionality of an application whereas connectors focus on the communication between components. Thus, each component defines application logic that is relatively independent of that provided by other components. A connector acts on behalf of components in terms of communication between components, encapsulating the details of inter-component communication.

An approach for designing secure software architectures is to encapsulate security capabilities within connectors separately from application components [8, 9]. The original role of connectors in the software architecture is to provide the mechanism for message communication between components [12]. However, in this paper, the role of connectors is extended to become secure connectors by adding security patterns [4, 7] to the connectors.

The secure connectors are designed using component-based concepts in which a secure connector is designed as a composite component that contains simple components that encapsulate the security patterns and the message communication pattern. One or more security pattern components are encapsulated in a secure connector to provide application components with one or more security services.

A. Design of Security Pattern Components

A security service is software functionality for realizing a security goal, such as authentication, authorization, confidentiality, integrity, availability or non-repudiation, which can be implemented by means of different security techniques. A security service can be realized by means of different security patterns, each of which addresses a specific security technique that realizes a security service. For instance, a confidentiality security service can be realized by means of a symmetric encryption security pattern [4] or an asymmetric encryption security pattern [4].

A security pattern is designed using one or two security pattern components (SPCs), as depicted in Fig. 1. The confidentiality security service can be realized using the symmetric encryption security pattern (Fig. 1a) [4], which is composed of the symmetric encryption encryptor and decryptor SPCs. The integrity security service can be realized with the hashing security pattern [4], which is designed with the hashing signer SPC (Fig. 1d) and hashing verifier SPC (Fig. 1d). The non-repudiation security service can be realized with the digital signature pattern [4], which is designed with the digital signature signer SPC (Fig. 1e) and digital signature verifier SPC (Fig. 1e). The authenticator and access control security patterns are realized respectively with the authenticator SPC (Fig. 1b) [4] and the authorization SPC (Fig. 1c) [4]. Each port of a component is defined in terms of provided and/or required interfaces [5]. Each security pattern component (Fig. 1) has a provided port through which the component provides security services to other components. Fig. 2 depicts the interfaces provided by the ports of the SPCs in Fig. 1.

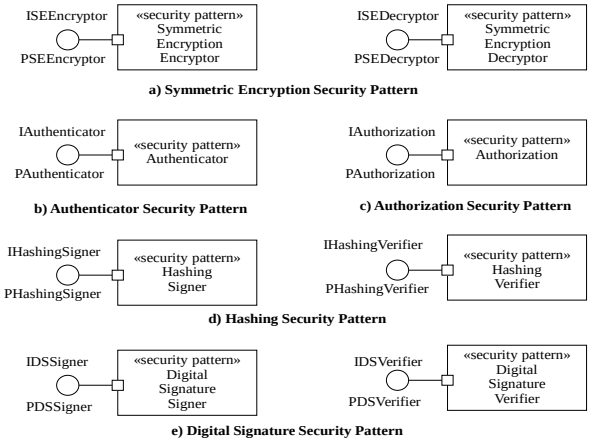


Fig. 1. Security Pattern Components

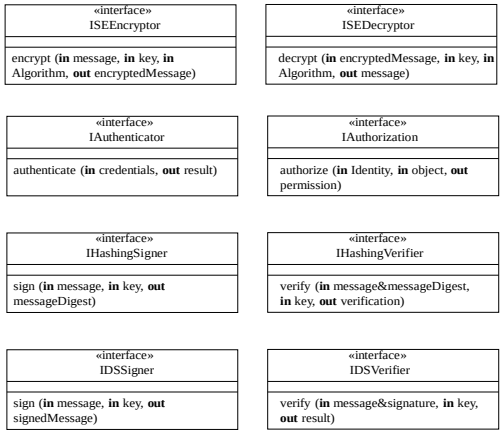
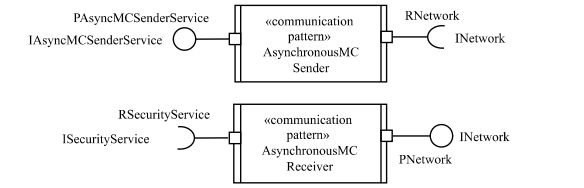


Fig. 2. Interfaces of Security Pattern Components

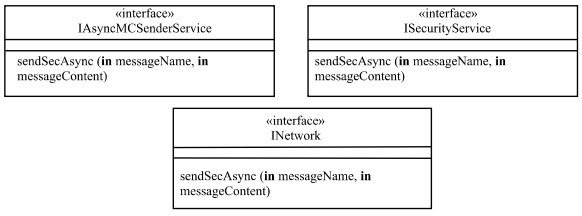
B. Design of Communication Pattern Components

Although there are other types of communications between distributed components, typical message communication patterns between the components are synchronous message communication with reply, synchronous message communication without reply, asynchronous message communication, and bidirectional asynchronous message communication [5]. Each communication pattern is designed with a sender communication pattern component (CPC) and a receiver communication pattern component (CPC), which are encapsulated in a secure sender connector and a secure receiver connector respectively.

In asynchronous message communication, an asynchronous message is sent from a sender component to a receiver component and is stored in a queue if the receiver is busy. The sender component can continue to send the next message to the receiver component as long as the queue is not full. Fig. 3a depicts the Asynchronous Message Communication (AMC) Sender CPC and Asynchronous Message Communication (AMC) Receiver CPC for the secure asynchronous message communication connector. The AMC Sender CPC (Fig. 3a) has the provided PAsyncMCSenderService port through which the Security Sender Coordinator component sends to the AMC Sender CPC a message being sent to the receiver application component, whereas it requests a service from the AMC Receiver CPC via the required RNetwork port. Similarly, the AMC Receiver CPC (Fig. 3a) has the required RSecurityService port and provided PNetwork port. Fig. 3b depicts the interfaces provided by each port of the AMC Sender and Receiver communication pattern components (CPCs).



a) Asynchronous Message Communication Sender and Receiver Communication Pattern Components



b) Interfaces of Asynchronous Message Communication Sender and Receiver Communication Pattern Components

Fig. 3. Asynchronous Message Communication Sender and Receiver Communication Pattern Components and their Interfaces

C. Design of Security Coordinators

A secure connector is designed by separately considering the message communication pattern and the security patterns required by application components. A secure connector is a distributed connector, which consists of a secure sender connector and a secure receiver connector that communicate with each other. Each secure connector is labeled with the UML stereotype «secure connector» to clearly identify its role in the software architecture. A secure sender or receiver connector consists of a security coordinator, one or more security objects, and a communication object [10].

A security coordinator, which is either a security sender coordinator or a security receiver coordinator, is designed for integrating the communication pattern and security patterns selected for a reusable secure connector. The security sender and receiver coordinators need to be designed for each reusable secure connector whenever a CPC and one or more SPCs are selected for the connector. In addition, a template for the high-level security coordinator can be designed for each communication pattern. The template is customized for each reusable secure connector based on the security pattern(s) selected.

Fig. 4 depicts the interfaces provided by the security sender coordinator (Fig. 6b) for a secure AMC connector. The senderSecurityPatternAttribute parameter in sendSecAsync() specifies the private key or secret key that is needed by security pattern components to apply security services to a message, or the algorithm that a security pattern component should select for a security service. The pseudocode template for the security sender coordinator is depicted in Fig. 5 in which the security related code (in italics) is replaced by the pseudocode for the security patterns selected for a secure AMC connector. Similarly, the pseudocode template for the Security Receiver Coordinator can be specified.

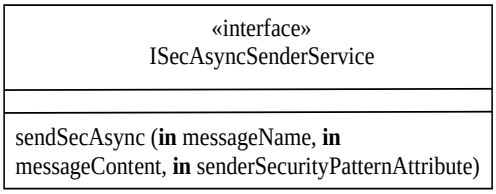


Fig. 4. Security Sender Coordinator Interfaces for Secure AMC Connector

```
loop

-- Wait for message from sender component;
receive (SenderComponentMessageQ, message);
Extract MessageName, MessageContent and
SenderSecurityPatternAttribute from message;

-- Apply security patterns to message content;
 while SecurityPatternsRequiredByMessageContent do
        Apply security pattern to message content;
end while;

-- Send message to AMC Sender CPC;
AsynchronousMCSender.sendSecAsync (in MessageName, in
MessageContent);

end loop;
```

Fig. 5. Pseudocode template for Security Sender Coordinator in Secure AMC Connector

The secure connectors are constructed by reusing SPCs (Fig. 1 and Fig. 2) and CPCs (Fig. 3) with the security coordinator (Fig. 6b) being the component that integrates the selected SPCs with the selected CPCs. Once one or more security patterns required by an application component are determined, the corresponding SPCs are selected from the reusable SPCs (Fig. 1 and Fig. 2). Similarly, the required CPCs are selected from the reusable CPCs (Fig. 3) in accordance with the communication pattern to be used between the application components. Also, for the selected CPCs, the corresponding security coordinator pseudocode template (Fig. 5) is customized to create the security coordinator component (Fig. 6) for the selected security pattern component(s). The security coordinator component integrates the selected SPCs with the selected CPC by sequencing the interaction with those components. For the integration, the security coordinator component has required ports through which it requests security services from the SPCs and communicates with the CPC. Also the security coordinator components provide ports for receiving a service request message from or requesting a service from an application component.

IV. Secure Asynchronous Message Communication Connector

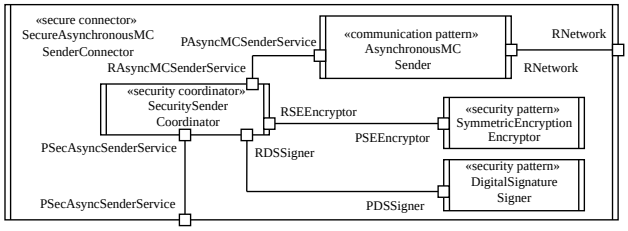
A. Design of Secure Asynchronous Message Communication Connector

Suppose that a secure AMC connector provides application components with the Symmetric Encryption and Digital Signature security patterns between the sender and receiver application components. This secure AMC connector is composed of a secure AMC sender connector (Fig. 6a) and a secure AMC receiver connector. The secure AMC sender connector (Fig. 6a) is designed as a composite component in which the Security Sender Coordinator component (Fig. 6b) integrates the reusable Symmetric Encryption Encryptor and Digital Signature Signer SPCs (Fig. 1) for the confidentiality and non-repudiation security with the reusable AMC Sender CPC (Fig. 3).

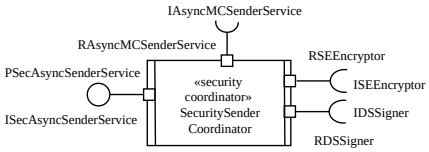
For integrating the components, the Security Sender Coordinator component (Fig. 6b) has a required RSEEncryptor port to communicate with a provided PSEEncryptor port of the Symmetric Encryption Encryptor SPC, which encrypts messages using the sender's secret key and algorithm selected by the sender component, and it also has a required RDSSigner port to communicate with a provided PDSSigner port of the Digital Signature Signer SPC, which signs a message using the sender's private key. The signed and encrypted messages are sent to the receiver component. The pseudocode for the Secure Sender Coordinator component is customized from the pseudocode template (Fig. 5) and is depicted in Fig. 7.

The Digital Signature Signer SPC depicted in Fig. 8 is a composite component that is composed of the Signer and Signature Algorithm security components. The Signer security component encrypts a message using an application sender component's private key by calling the Signature Algorithm security component to create a signature, and then creates a signed message that contains the original message and the signature. The private key is managed by the security sender coordinator on behalf of an application sender component. The Signer security component has the provided PDSSigner port to communicate with the provided PDSSigner port of the Digital Signature Signer SPC, and it also has the required RSignatureAlgorithm to communicate with the Signature Algorithm security component, which generates the signature of message. Fig. 8c depicts the interfaces for Signer and Signature Algorithm security components.

Similarly, the AMC Receiver Connector is designed as a composite component that encapsulates the Security Receiver Coordinator component, Symmetric Encryption Decryptor SPC, Digital Signature Verifier SPC, and AMC Receiver CPC.



a) Secure Asynchronous Message Communication Sender Connector



b) Security Sender Coordinator and its Interface

Fig. 6. Security Sender Coordinator and Secure Asynchronous Message Communication Sender Connector

```
loop

-- Wait for message from sender component;
receive (SenderComponentMessageQ, message);
Extract MessageName, MessageContent, PrivateKey, SecretKey, and Algorithm from message;

-- Apply security patterns to message content;
if MessageContent requires non-repudiation
then
        DigitalSignatureSigner.sign (in MessageContent, in PrivateKey,
        out SignedMessageContent);
        MessageContent = SignedMessageContent;
end if;
if MessageContent requires confidentiality
then
        SymmetricEncryptionEncryptor.encrypt (in MessageContent,
        in SecretKey, in Algorithm, out EncryptedMessageContent);
        Message Content = EncryptedMessageContent;
end if;

-- Send message to AMC Sender CPC;
AsynchronousMCSender.sendSecAsync (in MessageName, in MessageContent);

end loop;
```

Fig. 7. Security Sender Coordinator for Secure AMC Communication Pattern and Symmetric Encryption Encryptor & Digital Signature Security Patterns

B. Example of Secure Asynchronous Message Communication Connector

This section describes how a secure AMC connector can be reused for different applications if the applications require the same security services and asynchronous message

communication. Fig. 9 depicts the structural view of a reusable secure AMC connector with Symmetric Encryption security pattern and Digital Signature security pattern, which can be applied for confirming a shipment in a business to business (B2B) electronic commerce application. When a Supplier component sends a shipment confirmation to a Delivery Order Server, the shipment confirmation is signed by the Digital Signature Signer SPC in the secure AMC sender connector assuming the Supplier Component requires a non-repudiation security service. The shipment confirmation and signature is then encrypted by the Symmetric Encryption Encryptor SPC in the secure AMC sender connector assuming it also requires a confidentiality security service. The encrypted shipment confirmation and signature are decrypted by the Symmetric Encryption Decryptor SPC, and then sent to the Delivery Order Server via the secure AMC receiver connector, which requests the sender component's public key from the Public Key Repository that is a certificate authority for public key infrastructure. The signature is verified by the secure AMC receiver connector with the sender's public key. The behavioral view of a reusable secure AMC connector can be depicted using UML communication or sequence diagrams. An example is described in [11] for confidentiality and non-repudiation security services.

The secure AMC connector (Fig. 9) with Symmetric Encryption security pattern and Digital Signature security pattern can be reused for sending a purchase order in the B2C electronic commerce application as well. The B2C application is required for sending a purchase request from a customer component to a supplier component in the B2C electronic commerce application.

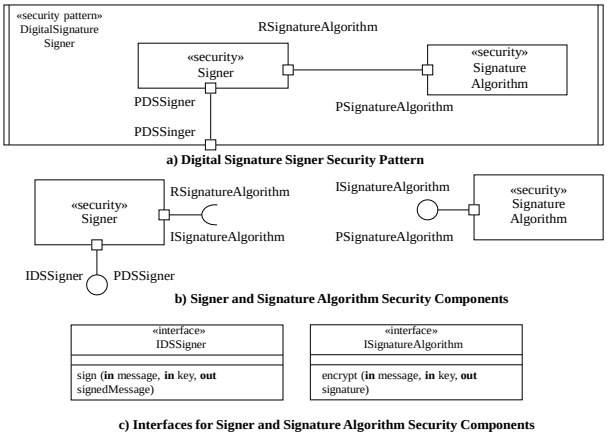


Fig. 8. Signer and Signature Algorithm Security Components in Digital Signature Signer Security Pattern Component and their Interfaces

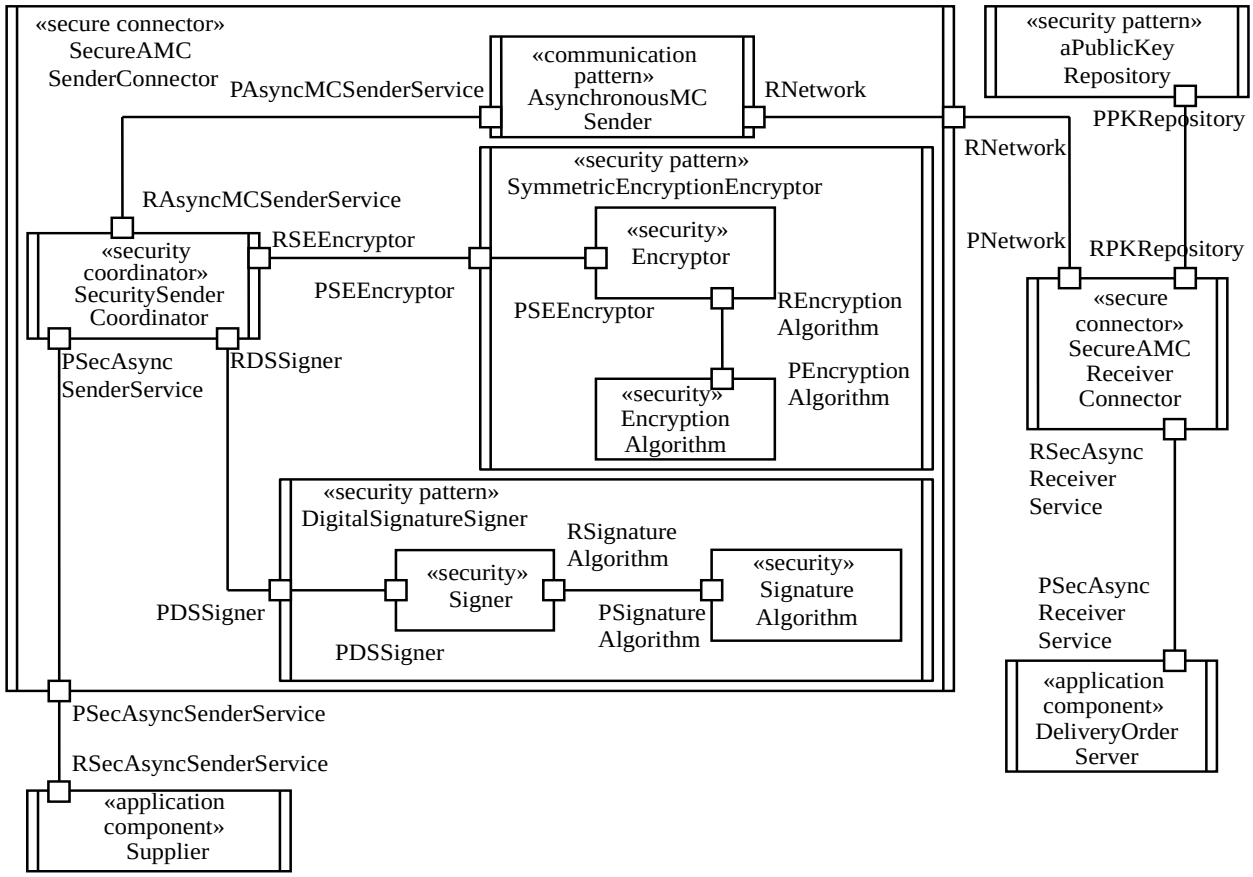


Fig. 9. Applying Secure AMC Connector with Symmetric Encryption and Digital Signature security patterns in B2B application

V. Validation of Reusable Secure Connectors

A. Reusability of Secure Connectors

The reusable secure connectors have been implemented using object-oriented programming in Java. The secure AMC connector was implemented with four threads for each of security sender coordinator component, AMC sender communication pattern component, security receiver coordinator component, and AMC receiver communication pattern component. A sender component and a receiver component for the secure AMC were implemented using each separate thread. Also, a message queue was implemented and placed between threads (for example, a message queue between the sender component thread and security sender coordinator component thread, and a message queue between the security sender coordinator component thread and the AMC sender communication pattern component thread). The secure AMC communication was implemented with 6 threads and 5 message buffers. Similarly, the secure synchronous message communication with reply (SMCWR) connector was implemented.

The implementation of the secure AMC connector and the secure SMCWR connector were applied to different applications. The secure asynchronous message communication connector was implemented and reused for Confirm Shipment (B2B) and Purchase Order (B2C) use cases. The secure SMCWR connector was implemented and reused for the ATM, B2B, and B2C applications, in particular to implement use cases such as PIN validation (ATM), Browse Catalog and Place Requisition (B2B), and Pay Product (B2C).

B. Performance Overhead of Reusable Secure Connectors

This section describes the performance analysis of secure applications using the secure connectors and compares them with secure applications executing the same message communication patterns but using other approaches for providing or not providing security. The three approaches compared in this section are the (1) *with secure connector* approach, for secure applications that use the approach described in this paper; (2) *without security pattern* approach, for applications that do not provide any security; (3) *without secure connector* approach, for secure applications in which security patterns are mingled with the application logic. The underlying difference between the *with secure connector* and *without secure connector* approaches is that the security patterns in the *without secure connector* approach are implemented within application components along with application logic, whereas the *with secure connector* approach has separated the security patterns from application components and implemented them as secure connectors. The *without security pattern* approach implements the application logic without any security patterns.

Table 1 shows the average time of multiple message communications and a comparison between the *with secure connector* approach, *without security pattern* approach, and *without secure connector* approach. For the *with secure connector* approach, the second column of Table 1 shows that the average communication time is 46.7 milliseconds (ms) for secure synchronous message communication with reply, and 57 ms for secure asynchronous message communication. Secure AMC takes more time than secure SMCWR because it consumes processing time for generating/verifying digital signatures in a public key infrastructure. The third and fourth columns of Table 1 show the average communication times for the corresponding patterns using the *without security pattern approach* and *without secure connector approach* respectively. The fifth column of Table 1 indicates that the time difference between the *with secure connector approach* and the *without security pattern* approach is highly significant. This is because *with secure connector* approach provides application components with security patterns such as confidentiality and non-repudiation. The security patterns in the *with secure connector* approach consume processing time for encrypting/decrypting messages and/or generating/verifying digital signatures, whereas the *without security pattern* approach is much faster due to it providing no security. Thus, the additional processing time taken by the *with secure connector* approach is to make applications secure in comparison to insecure applications developed using the *without security pattern* approach.

Comparing the performance of the *without secure connector* and *with secure connector* approaches shows that there is no significant difference in the runtime performance of the two approaches. The time difference between the two approaches (sixth column in Table 1) ranges from less than 0.6 ms to 1 ms. Both approaches provide applications with security patterns, but the *with secure connector* approach separates security patterns from application logic, so that it leads to secure software architectures that are more maintainable and evolvable than the *without secure connector* approach.

Table 1: Performance comparison of the *with secure connector* approach, *without security pattern* approach, and *without secure connector* approach

Communication pattern	With secure connector approach	Without security pattern approach	Without secure connector approach	Time difference between with secure connector approach and without security pattern approahc	Time difference between with secure connector approach and without secure connector approach
Synchronous Message Communication With Reply (SMCWR)	46.7 ms	17.8 ms	46.1 ms	28.9 ms	0.6 ms
Asynchronous Message Communication (AMC)	57.0 ms	8.9 ms	56.0 ms	48.1 ms	1.0 ms

VI. Conclusions

A secure connector can be reused in different applications if it matches the security and communication patterns required between application components. Reusable secure connectors

are designed as composite components using model-based CBSA concepts, which are designed by reusing security pattern components providing security services required by application components as well as communication pattern components for transmission of secure messages and responses between components. Integration of security and communication patterns within secure connectors is provided by security coordinators. In particular, this paper contributes to the reusable secure connectors by providing security patterns for security services, integrating security patterns and communication patterns within secure connectors, and by providing a pseudocode template for the high-level security coordinator that can be customized for each secure connector. To validate this approach, secure connectors were implemented for two electronic commerce applications and an ATM application. A performance evaluation was also carried out comparing approaches with and without secure connectors.

This paragraph describes future research for secure connectors. Component-based secure connectors might be mapped to aspect-oriented secure connectors [2] by considering the relationships between the ports/interfaces of components and the pointcuts/advices of aspects. Future work also includes extending the reusable secure connector approach to incorporate additional communication and security patterns.

ACKNOWLEDGMENT

Gomaa's research is supported by the Air Force Office of Scientific Research under grant number FA9550-16-1-0030.

REFERENCES

[1] Al-Azzani, S., Bahsoon, R., 2012. SecArch: Architecture-level Evaluation and Testing for Security, Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), Joint Working IEEE/IFIP Conference on, August.

[2] Baker, C., Shin M., 2014. Aspect-Oriented Secure Connectors for Implementation of Secure Software Architecture, International Conference on Software Engineering and Knowledge Engineering (SEKE'2014), Vancouver, Canada, July 1-3.

[3] Basin, D., Clavel, M., Egea, M., 2011. A Decade of Model-Driven Security, 16th ACM symposium on Access control models and technologies, Innsbruck, Austria, June 15-17.

[4] Fernandez, E. B., 2013. Security Patterns in Practice, Wiley.

[5] Gomaa, H., 2011. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures, Cambridge University Press.

[6] Ren, J., Taylor, R., Dourish, P., Redmiles, D., 2005. Towards An Architectural Treatment of Software Security: A Connector-Centric Approach, Proceedings of the Workshop on Software Engineering for Secure Systems, St. Louis, MI, USA, May .

[7] Schumacher, M., Fernandez, E. B., Hybertson, D., Buschmann, F., Sommerlad, P., 2006. Security Patterns, Wiley.

[8] Shin, M. E., Gomaa, H., 2007. Software Modeling of Evolution to a Secure Application: From Requirements Model to Software Architecture, Science of Computer Programming, Volume 66, Issue 1, pp. 60-70.

[9] Shin, M. E., Malhotra, B., Gomaa, H., Kang, T., 2012. Connectors for Secure Software Architectures, 24th International Conference on Software Engineering and Knowledge Engineering (SEKE'2012), San Francisco, July 1-3.

[10] Shin, M. E., Gomaa, H., Pathirage, D., 2016. Reusable Secure Connectors for Secure Software Architecture, 15th International Conference on Software Reuse, June 2016, Limassol, Cyprus.

[11] Shin, M. E., Gomaa, H., Pathirage, D., Baker, C., Malhotra, B., 2016. Design of Secure Software Architectures with Secure Connectors, International Journal of Software Engineering and Knowledge Engineering, Vol. 26, No. 5, pp 769–805.

[12] Taylor, R. N., Medvidovic, N., Dashofy, E. M., 2010. Software Architecture: Foundations, Theory, and Practice, Wiley & Sons.

[13] Lodderstedt, T., Basin, D., Doser, J., 2002. SecureUML: A UML-Based Modeling Language for Model-Driven Security, Intl. Conf. on the Unified Modeling Language, London, UK.

[14] Jürjens, J., 2002. UMLsec: Extending UML for Secure Systems Development, Fifth International Conference on the Unified Modeling Language, London, UK.

[15] Gomaa, H., Menasce, D. A., Shin, M. E., 2001. Reusable Component Patterns for Distributed Software Architectures, Proceedings of ACM Symposium on Software Reusability, ACM Press, Pages 69-77, Toronto, Canada, May.

[16] Rumbaugh, J., Booch, G., Jacobson, I., 2004. The Unified Modeling Language Reference Manual, Addison-Wesley.

[17] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D. A. Menascé, "Software Adaptation Patterns for Service-oriented Architectures," Proceedings ACM Symposium on Applied Computing, Sierre, Switzerland, March 2010.

[18] E. Albassam, H. Gomaa, and D. A. Menascé, "Model-based Recovery Connectors for Self-adaptation and Self-healing: Design and Experimentation," in Software Technologies, Revised Selected Papers from the 11th International Joint Conference, ICSOFT 2016, Lisbon, Portugal, July, 2016; Published by Springer, CCIS 743 pp. 108-131, 2017.