Design of Adaptable and Secure Connectors for Software Architectures

Juan Marcelo Gutierrez Carballo[1], Michael Shin[1] and Hassan Gomaa[2]
[1]*Department of Computer Science, Texas Tech University, Lubbock, TX, U.S.A.*
[2]*Department of Computer Science, George Mason University, Fairfax, VA, U.S.A.*

Keywords: Adaptable Connector, Secure Connector, Adaptable Secure Software Architecture, Component-Based Software Architecture, Asynchronous Message Communication.

Abstract: This paper describes the design of adaptable and secure (AS) connectors that encapsulate security concerns and their adaptation concerns in the interaction between application components in secure software architectures. The security concerns in software architectures need to be dynamically adaptable to changing security requirements so that the architectures respond to evolving security risks. This paper describes the design of AS connectors that dynamically adapt security concerns over changing security risks in software architectures. The AS connectors can reduce the complexity of adaptation separately from application concerns. To validate this research, we designed and implemented the AS connectors for asynchronous message communication (AMC), which would adapt security concerns for secure software architectures to changing security risks.

1 INTRODUCTION

Secure software architectures are fundamental structures for secure applications. Increasingly, secure software architectures have been designed to develop secure application software. Secure software architectures address security and application concerns. However, the designs of these architectures typically mixes security and application concerns, which increases the complexity of the architectures. To cope with this, secure connectors (Shin et al., 2016a, 2016b, 2017, 2018, 2019, 2021) were designed to separate security concerns for interactions between application components from application concerns to decrease the complexity of secure software architectures.

However, the secure connectors (Shin et al., 2016a, 2016b, 2017, 2018, 2019, 2021) do not address the adaptation of security concerns between application components in secure software architectures. The security concerns for secure software architectures need to be adapted at runtime so that the architectures respond to evolving security risks. Dynamic adaptation of security concerns in software architectures makes the architectures more complex. The current secure connectors do not enable secure software architectures to evolve to changing security requirements. Therefore, it is necessary for an approach to designing secure connectors that are adaptable to evolving security in secure software architectures, which can reduce the complexity of dynamic adaptation for the architectures.

This paper designs adaptable and secure (AS) connectors that evolve secure software architectures adaptable to changing security requirements. AS connectors encapsulate security concerns for the interaction between application components as well as the adaptation logic of the security concerns. AS connectors enable secure software architecture to evolve security at runtime for message communication between application components. AS connectors are designed by extending secure connectors (Shin et al., 2016a, 2016b, 2017, 2018, 2019, 2021) to adaptable connectors.

The structure of this paper is as follows. Section 2 describes related work. Section 3 presents secure connectors. Section 4 describes the design of AS connectors, followed by validation of this research in section 5. Section 6 concludes this paper with future research.

2 RELATED WORK

Related work explores existing literature and methodologies related to the adaptability of software

Gutierrez Carballo, J., Shin, M. and Gomaa, H.
Design of Adaptable and Secure Connectors for Software Architectures.
DOI: 10.5220/0012856900003753
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 19th International Conference on Software Technologies (ICSOFT 2024)*, pages 185-192
ISBN: 978-989-758-706-1; ISSN: 2184-2833
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

systems aligning with changing requirements including security.

The study (Gottschalk, Yigitbas, and Engels, 2022) presents a model-driven framework for continuous experimentation on component-based software architectures, emphasizing dynamic adaptability but not security. The study (Vogel, 2018) introduces mRUBiS, a framework for model-based self-adaptation and optimization. The author (Huynh, 2019) discusses the importance of state transfer management in adaptive software for maintaining consistency during dynamic adaptation.

Authors (Gao et al., 2021) propose a service-oriented dynamic and adaptive software architecture, focusing on service component reuse and optimization. The study (Mutanu and Kotonya, 2019) delves into runtime adaptation in service-oriented systems but lacks a focus on security adaptation. The authors (De Sanctis, Bucchiarone, and Marconi, 2020) advocate for adaptivity from the earliest stages of service-based applications, supporting continuous integration of new services.

The research (De Sanctis, Muccini, and Vaidhyanathan, 2020) explores the intricacies of data-driven adaptation in microservice-based IoT architectures, managing adaptation in resource-constrained environments. The study (Shin, Nejati, Sabetzadeh, Briand, Arora, and Zimmer, 2020) introduces a dynamic adaptive congestion control algorithm for software-defined networks (SDNs) in IoT systems, enhancing performance and reliability.

The authors (Philip et al., 2020) propose a composite software architectural style for flexibility and dynamic adaptation but do not address security concerns. The study (Mayrhofer et al., 2019) assesses adaptability in software architectures for Cyber-Physical Production Systems, emphasizing runtime adaptability in manufacturing.

The research (Chen, 2019) explores retrained versus incremental machine learning models for performance prediction in adaptable systems, providing insights into flexibility and accuracy. Our approach embeds adaptive security measures within connectors. The research (Kaya et al., 2019) discusses runtime adaptability in Ambient Intelligence systems using a component-oriented approach.

The research (Shin et al., 2016a, 2016b, 2017, 2018, 2019, 2021) designs secure connectors for distributed component-based software architectures, integrating communication and security patterns. Our research in this paper focuses on runtime security adaptation with AS connectors. A co-author of this paper studied software adaptation for service-oriented systems (Gomaa et al., 2010; Gomaa and Hashimoto, 2012), dynamically adapting service-oriented architectures using adaptation connectors. Compared to the study, our research proposes adaptable and secure connectors to adapt the software architectures to changing security requirements.

Authors (Porter and Albassam, 2020) investigate decentralized architecture-based self-protection for adaptable security systems.

3 SECURE CONNECTORS

A secure connector is a distributed connector consisting of a secure sender connector and a secure receiver connector that communicate with each other. A secure sender or receiver connector is designed with a security coordinator, zero or more security pattern components (SPCs), and one or more communication pattern components (CPCs).

3.1 Security Pattern Components

A security pattern addresses a solution to recurring security problems using a security mechanism against a security threat. A security pattern is designed with security pattern components (SPCs), as depicted in Fig. 1. For instance, confidentiality can be realized using the symmetric encryption security pattern (Fig. 1a) composed of the symmetric encryption encryptor and decryptor SPCs with their interfaces. The digital signature (Fig. 1b) is designed as the digital signature signer and digital signature verifier SPCs. Each port of a component is defined in terms of provided and/or required interfaces (Gomaa, 2011). Each security pattern component (Fig. 1) has a provided port through which the component provides security services to other components. Fig. 1c depicts the interfaces provided by the ports of the SPCs.

3.2 Communication Pattern Components

Each communication pattern is designed as 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. Fig. 2a depicts the asynchronous message communication (AMC) sender and receiver CPCs for the secure AMC connector. The AMC sender CPC (Fig. 2a) has the provided PAMCSenderService port through which it receives from the security sender coordinator component a message to be sent to the AMC receiver CPC via the required RNetwork port. Similarly, the

AMC receiver CPC (Fig. 2a) has the required RSecurityService port and provided PNetwork port. Fig. 2b depicts the interfaces provided by each port of the AMC sender and receiver CPCs.

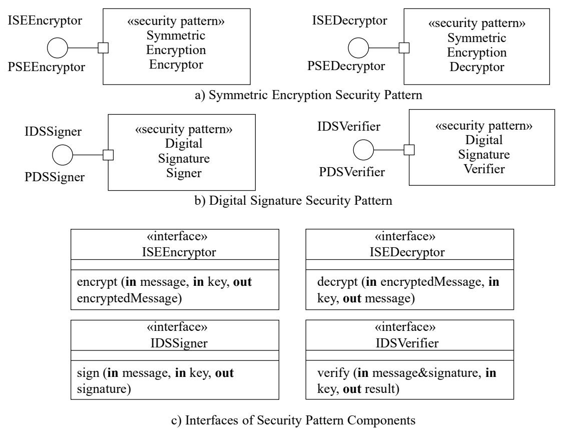
ISEEncryptor
PSEEncryptor
«security pattern» Symmetric Encryption Encryptor
ISEDecryptor
PSEDecryptor
«security pattern» Symmetric Encryption Decryptor
a) Symmetric Encryption Security Pattern
IDSSigner
PDSSigner
«security pattern» Digital Signature Signer
IDSVerifier
PDSVerifier
«security pattern» Digital Signature Verifier
b) Digital Signature Security Pattern
«interface» ISEEncryptor
encrypt (**in** message, **in** key, **out** encryptedMessage)
«interface» ISEDecryptor
decrypt (**in** encryptedMessage, **in** key, **out** message)
«interface» IDSSigner
sign (**in** message, **in** key, **out** signature)
«interface» IDSVerifier
verify (**in** message&signature, **in** key, **out** result)
c) Interfaces of Security Pattern Components

Figure 1: Security pattern components and their interfaces.

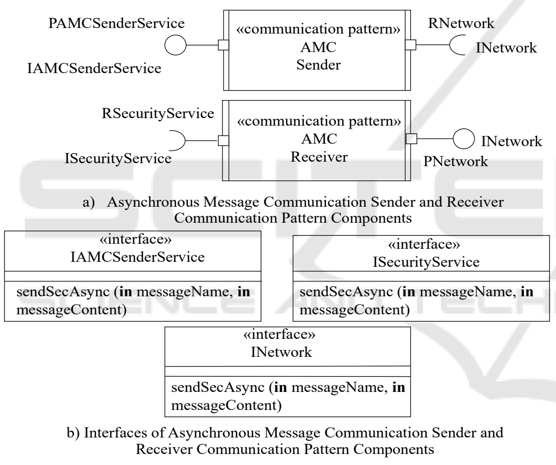
PAMCSenderService
IAMCSenderService
«communication pattern» AMC Sender
RNetwork
INetwork
RSecurityService
ISecurityService
«communication pattern» AMC Receiver
INetwork
PNetwork
a) Asynchronous Message Communication Sender and Receiver Communication Pattern Components
«interface» IAMCSenderService
sendSecAsync (**in** messageName, **in** messageContent)
«interface» ISecurityService
sendSecAsync (**in** messageName, **in** messageContent)
«interface» INetwork
sendSecAsync (**in** messageName, **in** messageContent)
b) Interfaces of Asynchronous Message Communication Sender and Receiver Communication Pattern Components

Figure 2: Asynchronous Message Communication Sender and Receiver Communication Pattern Components and their Interfaces.

3.3 Security Coordinator Components

A security coordinator, which is either a security sender coordinator or a security receiver coordinator, is designed to integrate the communication patterns and security patterns selected for a secure connector. The security sender and receiver coordinators need to be designed for each secure connector whenever one or more CPCs and zero or more SPCs are selected for the connector. A template (Shin et al., 2018, 2019) for the high-level security coordinator can be designed for each communication pattern. The template is customized for each secure connector based on the security pattern(s) selected.

4 DESIGN OF ADAPTABLE AND SECURE CONNECTORS

Adaptable and secure (AS) connectors are designed to adapt the SPCs encapsulated in the connectors as well as provide security services to application components. The AS connectors can be adapted at runtime by adding, removing, or replacing SPCs with others, according to changing security requirements.

The AS connectors are designed by extending secure connectors to adaptation. Figure 3 shows a high-level state machine view of an AS Connector. An AS connector works like a secure connector in an Active state (Fig. 3). Whereas, in an adaptation state (Fig. 3), it adapts security patterns at runtime according to changing security requirements. To prepare the adaptation of an AS connector, the Active state transitions to the Adaptation state (Fig. 3), which then transitions back to the Active state to reactivate the connector after adaptation (Fig. 3). The detailed statecharts for the AMC Sender Connector and the AMC Receiver Connector are depicted in Figures 6 and 7 below.

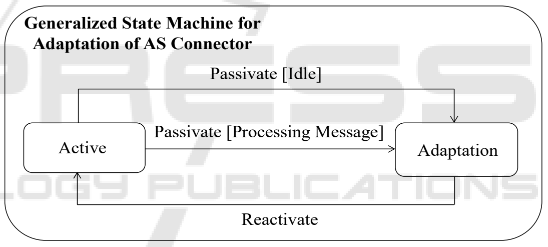
Generalized State Machine for Adaptation of AS Connector
Passivate [Idle]
Active
Passivate [Processing Message]
Adaptation
Reactivate

Figure 3: Statechart for Security Coordinator in Adaptable and Secure Connector.

An AS connector is designed as an AS sender connector and an AS receiver connector, each with zero or more security pattern components (SPCs), one or more communication pattern components (CPCs), and a security coordinator. Fig. 4a depicts an AS asynchronous message communication (AMC) sender connector with a symmetric encryption encryptor SPC, which sends a message from one application component to another. The AS AMC sender connector (Fig. 4a) is designed as a composite component. The security sender coordinator component (Fig. 4b) is designed with ports to integrate the Symmetric Encryption Encryptor SPC, Key SPC, and AMC sender CPC, allowing it to receive a message or adaptation command. An application component sends a message to the Security Sender Coordinator through the provided PSecAsyncSenderService port. The message is encrypted by the Symmetric Encryption Encryptor

SPC using the key retrieved from the Key SPC and sent by the AMC sender CPC to an AS AMC receiver connector through the required RNetwork port. Fig. 4b also shows the interfaces of the security sender coordinator component.

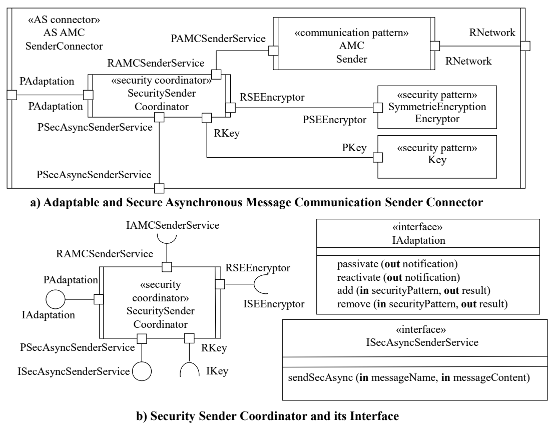
«AS connector» AS AMC SenderConnector
PAMCSenderService
«communication pattern» AMC Sender
RNetwork
RNetwork
RAMCSenderService
PAdaptation
PAdaptation
«security coordinator» SecuritySender Coordinator
RSEEncryptor
«security pattern» SymmetricEncryption Encryptor
PSEEncryptor
PSecAsyncSenderService
RKey
PKey
«security pattern» Key
PSecAsyncSenderService
a) Adaptable and Secure Asynchronous Message Communication Sender Connector
IAMCSenderService
RAMCSenderService
PAdaptation
IAdaptation
«security coordinator» SecuritySender Coordinator
RSEEncryptor
ISEEncryptor
PSecAsyncSenderService
ISecAsyncSenderService
RKey
IKey
«interface» IAdaptation
passivate (out notification)
reactivate (out notification)
add (in securityPattern, out result)
remove (in securityPattern, out result)
«interface» ISecAsyncSenderService
sendSecAsync (in messageName, in messageContent)
b) Security Sender Coordinator and its Interface

Figure 4: Adaptable and Secure Asynchronous Message Communication Sender Connector and Security Sender Coordinator and its interfaces.

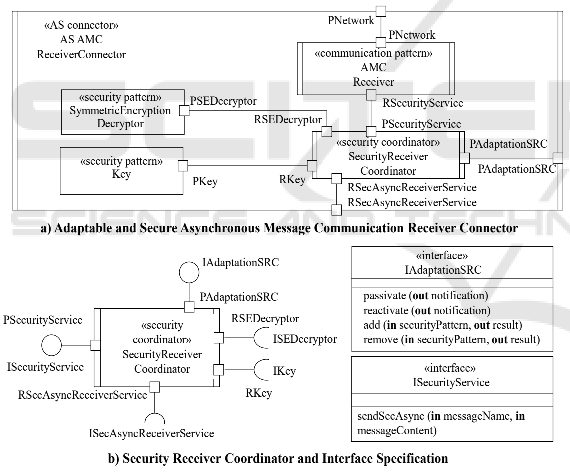
PNetwork
«AS connector» AS AMC ReceiverConnector
PNetwork
«communication pattern» AMC Receiver
«security pattern» SymmetricEncryption Decryptor
PSEDecryptor
RSEDecryptor
RSecurityService
PSecurityService
«security coordinator» SecurityReceiver Coordinator
PAdaptationSRC
PAdaptationSRC
«security pattern» Key
PKey
RKey
RSecAsyncReceiverService
RSecAsyncReceiverService
a) Adaptable and Secure Asynchronous Message Communication Receiver Connector
IAdaptationSRC
PAdaptationSRC
PSecurityService
ISecurityService
«security coordinator» SecurityReceiver Coordinator
RSEDecryptor
ISEDecryptor
IKey
RKey
RSecAsyncReceiverService
ISecAsyncReceiverService
«interface» IAdaptationSRC
passivate (out notification)
reactivate (out notification)
add (in securityPattern, out result)
remove (in securityPattern, out result)
«interface» ISecurityService
sendSecAsync (in messageName, in messageContent)
b) Security Receiver Coordinator and Interface Specification

Figure 5: Adaptable and Secure Asynchronous Message Communication Receiver Connector and Security Receiver Coordinator and its interfaces.

An AS asynchronous message communication (AMC) receiver connector (Fig. 5a) is designed to receive a message for a receiver application component from an AS AMC sender connector (Fig. 4a). The security receiver coordinator component (Fig. 5b) in the AS AMC receiver connector (Fig. 5a) is designed with the required RSEDecryptor port to request the Symmetric Encryption Decryptor SPC to decrypt an encrypted message, the required RKey port to read a key from the Key SPC, and the required RSecAsynReceiverService port to forward a message to an application component. Fig. 5b depicts the interfaces of the security receiver coordinator component.

Adaptation of AS connectors is designed with security sender and receiver coordinator components. Fig. 6 depicts the design of the security sender coordinator (Fig. 4b) of the AS AMC sender connector (Fig. 4a) using the state machine, where the active composite state is composed of the Waiting For Message state that waits for the application component's message, the Retrieving Secret Key state that reads a secret key from the Key SPC (Fig. 4a), and the Encrypting state in which the Symmetric Encryption Encryptor (Fig. 4a) encrypts the message.

In the active composite state (Fig. 6), a message arrived at the security sender coordinator in the Waiting For Message state (Fig. 6) causes a transition to either the Retrieving Secret Key state (Fig. 6) if the secret key is unavailable or the Encrypting state if the key is available. As a message arrives, the message queue increases by one on those state transitions, which is modeled with q1++. An additional message arrival also increases the message queue by one while the state machine is either in the Retrieving Secret Key or Encrypting state, in which the queue increase is modeled as a self-transition with "Message/ q1++" (Fig. 6). The coordinator continues to encrypt messages in the Encrypting state while the messages are available (i.e., q1>1 in Fig. 6), decreasing the number of messages in the queue one by one (i.e., q1-- in Fig. 6). When the last message stored in the message queue (i.e., q1=1 in Fig. 6) is encrypted and sent to the AS AMC receiver connector, the state machine makes a transition to the Waiting For Message state.

The AS AMC sender connector makes a transition from the active state composite to the adaptation composite state, where the security sender coordinator is passivated and then becomes quiescent (Kramer and Magee, 1990). The adaptation state (Fig. 6) consists of the Passivating state (Kramer and Magee, 1990) completely sending the ongoing message to the AS AMC receiver connector for adaptation, and the Quiescent state (Kramer and Magee, 1990, 2007) adapting the SPCs. The Passivating state ensures that the ongoing message is delivered to the AS AMC receiver connector before the start of adaptation. The Passivating state is to maintain the consistency of connectors before and after the adaptation. The security sender connector (Fig. 4) completes retrieving a secret key or encrypting the ongoing message while its security sender coordinator is passivating.

In the adaptation composite state (Fig. 6), only the ongoing message is completely encrypted, and the encrypted message is sent to the AS AMC receiver connector. Whereas incoming messages are stored in the queue, which is modeled with a self-transition to the adaptation composite state (i.e., Message/ q1++ in Fig. 6). The incoming messages are not encrypted and stored in the queue, whereas encrypted messages are sent to the AS AMC receiver connector to keep consistency before and after adaptation. The messages stored in the queue are processed when the AS AMC sender coordinator is reactivated.

Similar to the security sender coordinator (Fig. 4b), The security receiver coordinator (Fig. 5b) of the AS AMC receiver connector (Fig. 5a) is modeled with active and adaptation composite states using the state machine (Fig. 7). The Active composite state is composed of the Waiting For Encrypted Message state waiting encrypted messages from the AMC receiver CPC, the Retrieving Secret Key state reading a secret key from the Key SPC, and the Decrypting state decrypting an encrypted message using the secret key. The adaptation composite state is modeled with the Passivating (Fig. 7) and Quiescent states for adapting SCPs.

In the Active composite state (Fig. 7), when a new encrypted message arrives in the Waiting For Encrypted Message state, the state machine (Fig. 7) transitions to either the Retrieving Secret Key state if the security receiver coordinator has a secret key or the Decrypting state if the key is available. q2++ in these state transitions means that the number of incoming encrypted messages increases by one as a new encrypted message arrives. Also, the number of encrypted messages increases by one for an additional encrypted message arrival while the state machine is either in the Retrieving Secret Key or Decrypting state. That is modeled as a self-transition with “Encrypted Message/ q2++” at the states (Fig. 7). The encrypted messages sent by the AS AMC sender connector is stored in a queue in the AS AMC receiver connector because this is asynchronous message communication. The coordinator continues to decrypt the encrypted messages in the Decrypting state if the encrypted messages are available (i.e., q2>1 in Fig. 7), decreasing the number of encrypted messages one by one (i.e., q2-- in Fig. 7). If the ongoing encrypted message is the last one (i.e., q2=1 in Fig. 7), the state machine transitions to the Waiting For Encrypted Message state. For adapting the AS AMC receiver connector, the security receiver coordinator is passivated by transitioning from the active composite state to the composite adaptation state.

The adaptation is performed to add to or remove SPCs from the AS sender and receiver connector when the security sender and receiver coordinators are in a Quiescent state. When the security coordinators are in a Quiescent state, the message delivery between application components is stopped temporarily until an adaptation has been completed. The security sender and receiver coordinators resume sending or receiving the messages when they are reactivated.

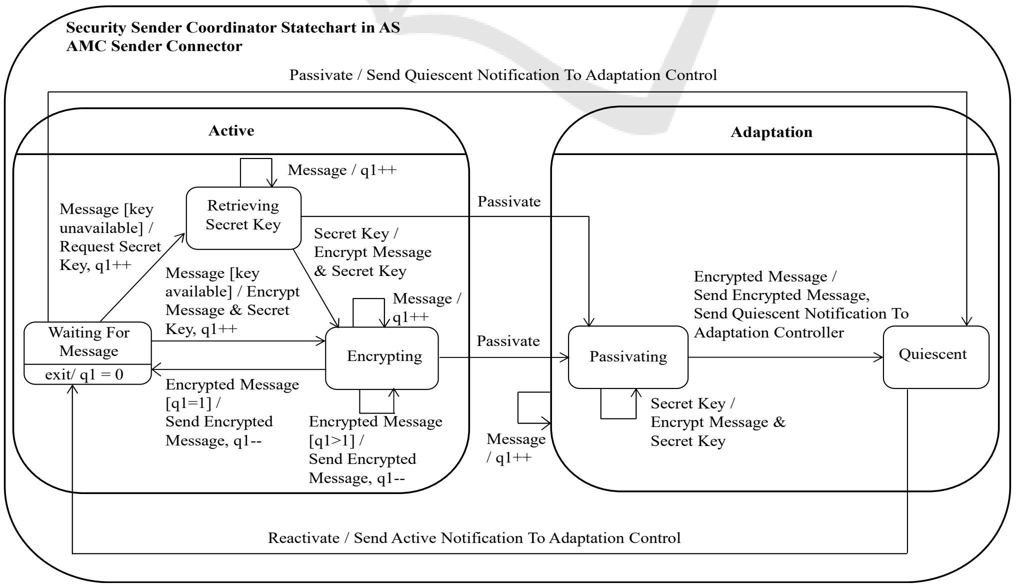
Security Sender Coordinator Statechart in AS AMC Sender Connector
Passivate / Send Quiescent Notification To Adaptation Control
Active
Adaptation
Message / q1++
Retrieving Secret Key
Message [key unavailable] / Request Secret Key, q1++
Secret Key / Encrypt Message & Secret Key
Passivate
Message [key available] / Encrypt Message & Secret Key, q1++
Message / q1++
Encrypted Message / Send Encrypted Message, Send Quiescent Notification To Adaptation Controller
Waiting For Message
exit/ q1 = 0
Encrypting
Passivate
Passivating
Quiescent
Encrypted Message [q1=1] / Send Encrypted Message, q1--
Encrypted Message [q1>1] / Send Encrypted Message, q1--
Message / q1++
Secret Key / Encrypt Message & Secret Key
Reactivate / Send Active Notification To Adaptation Control

Figure 6: Security Sender Coordinator Statechart for AS AMC Sender Connector with Symmetric Encryption Encryptor Security Pattern Component.

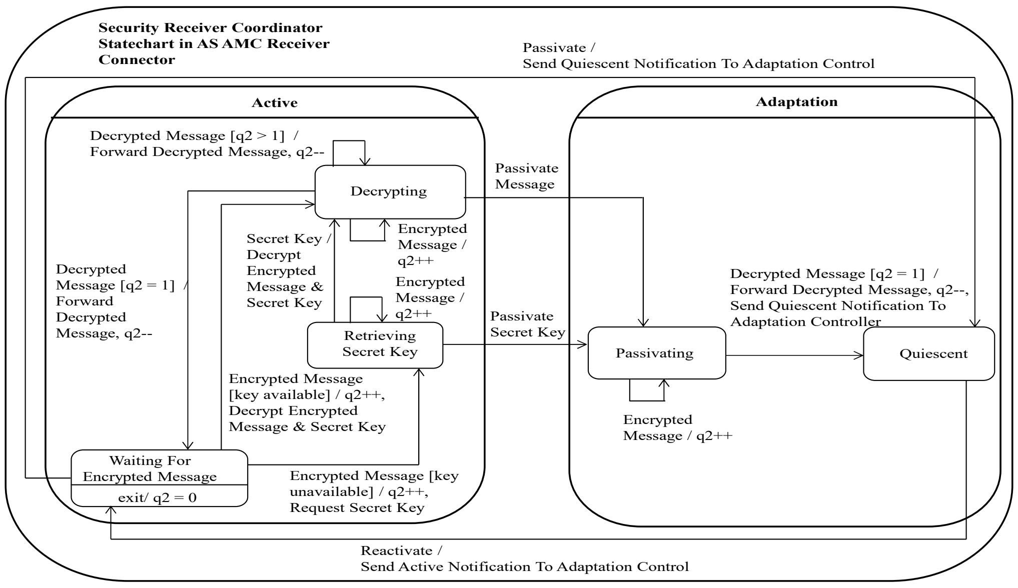
Security Receiver Coordinator Statechart in AS AMC Receiver Connector
Passivate / Send Quiescent Notification To Adaptation Control
Active
Adaptation
Decrypted Message [q2 > 1] / Forward Decrypted Message, q2--
Decrypting
Passivate Message
Encrypted Message / q2++
Secret Key / Decrypt Encrypted Message & Secret Key
Decrypted Message [q2 = 1] / Forward Decrypted Message, q2--
Encrypted Message / q2++
Retrieving Secret Key
Passivate Secret Key
Decrypted Message [q2 = 1] / Forward Decrypted Message, q2--, Send Quiescent Notification To Adaptation Controller
Passivating
Quiescent
Encrypted Message / q2++
Encrypted Message [key available] / q2++, Decrypt Encrypted Message & Secret Key
Waiting For Encrypted Message
exit/ q2 = 0
Encrypted Message [key unavailable] / q2++, Request Secret Key
Reactivate / Send Active Notification To Adaptation Control

Figure 7: Security Receiver Coordinator Statechart for AS AMC Receiver Connector with Symmetric Encryption Decryptor Security Pattern Component.

5 VALIDATION

5.1 Adaptation Architecture

The adaptation architecture is designed to implement the AS AMC connectors to validate this research. The adaptation architecture consists of the adaptation layer and the application layer. The adaptation layer controls the AS AMC connectors by sending adaptation commands (i.e., passivate or reactivate). The adaptation control component receives an adaptation request from an administrator (message sequence A0 in Fig. 8) and takes the necessary actions to align AS AMC connectors with the request. The application layer contains AS AMC connectors that securely communicate messages and adapt security patterns (Fig. 8).

Fig. 8 depicts the passivation, adaptation, and reactivation for adapting security pattern components in AS AMC connectors. The adaptation control initiates passivation, sending passivate commands to the security sender and receiver coordinators. When the AS sender and receiver connectors are quiescent, the adaptation control sends adaptation commands with SPC information to change existing SPCs encapsulated in the connectors. While adaptation is in progress, the message delivery requests from the sender component are queued until the connectors are reactivated. Upon reactivation, the AS AMC sender connector processes and sends the queued messages to the receiver component through the AS AMC receiver connector.

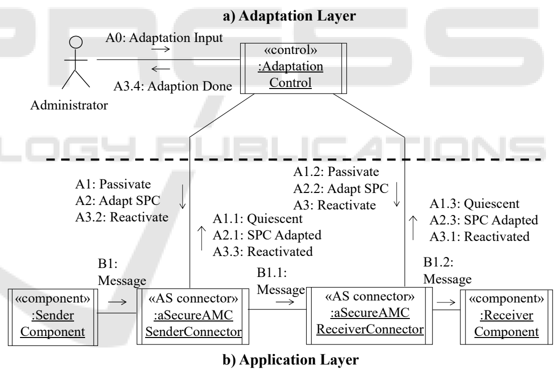

Figure 8: Adaptation Architecture for Adapting AS Sender and Receiver Connectors.

5.2 Implementation of Adaptable and Secure Connectors

The AS AMC sender and receiver connectors have been implemented in Java using multithread programming. Message queues were used to interact between the AS sender and receiver connectors in AS asynchronous message communication (AMC). Each connector uses multi-threads to send messages from the sender to the receiver component, with security sender and receiver coordinators invoking security pattern components (SPCs) to secure messages. Communication pattern components (CPCs) handle message transmission and reception. A listener thread awaits user input, invoking functions for adaptation control or modifying a global variable to control connector adaptation. A global flag variable is used to ensure that the connectors have completed their tasks before entering the quiescent state. The adaptation control sends SPC instances to the connectors. Removal of security patterns sends a null instance to decouple SPCs.

5.3 Testing AS Connectors

Various security pattern components were adapted to test AS AMC connectors while communicating messages. The sender application component sent messages every five seconds, and the receiver application component received them.

5.3.1 Initialization and Execution for Testing

- Initial Setup: The sender and receiver components are declared, and the queue sizes are set up to mimic real-world scenarios.
- Message Simulation: The sender component sent messages at fixed intervals to the AS AMC sender connector, and the sent messages were stored in a queue if the connector was in a passivating or quiescent state.
- Adaptation Control Logic: An input thread within the adaptation control awaited user input for passivating, adapting, or reactivating the AS AMC connectors.

5.3.2 Adapting AS Connectors and Observing Changes

- Passivation and Activation: Commands such as 'pas' (passivate) and 'act' (activate) controlled the connectors' operational state.
- Adding or Removing Security Pattern Components: SPCs were dynamically added or removed using the commands starting with 'add' or 'rem'. For instance, adding 'ee' activated the Symmetric Encryption Encryptor SPC in the AS AMC sender connector and the Symmetric Encryption Decryptor SPC in the AS AMC receiver connector.
- Runtime Monitoring: All adaptation steps were displayed on the adaptation commands to verify the design of the AS AMC connectors.

6 CONCLUSIONS AND FUTURE WORK

This paper has described the design of adaptable and secure (AS) connectors that encapsulate security concerns and their adaptation concerns in the interaction between application components in secure software architectures. To adapt the security concerns dynamically to changing security risks, we designed AS connectors using the state machines, which modeled the AS sender and receiver connectors with active and adaptation states. The AS sender and receiver connectors have been adapted only at quiescent states to maintain consistency before and after adaptation. To validate this research, we implemented the AS AMC connector and tested the adaptability of the connector at runtime.

We leave several future works to extend this research. This research could be extended with additional validation by implementing secure software architectures for Web-based or App-based applications with AS connectors. We can also develop a prototype tool that automatically generates the code for AS connectors to save the effort to implement each AS connector for different communication patterns. As another direction, we could integrate this research with a recovery mechanism to assist in the recovery of secure software systems from security failures. In addition, we could investigate developing an adaptation and recovery framework (Albassam, 2017) for secure software architectures designed with AS connectors.

REFERENCES

Albassam, E., Gomaa, H., Menasce, D., and Porter, P. (2017). DARE: A Distributed Adaptation and Failure Recovery Framework for Software Architectures. *Proceedings 14th IEEE International Conference on Autonomic Computing and Communications (ICAC)*, Columbus, Ohio.

Chen, T. (2019). All Versus One: An Empirical Comparison on Retrained and Incremental Machine Learning for Modeling Performance of Adaptable Software. *In SEAMS '19, Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, pp. 157–168.

De Sanctis, M., Bucchiarone, A., Marconi, A. (2020). Dynamic Adaptation of Service-Based Applications: A Design for Adaptation Approach. *In J Internet Serv Appl*, vol. 11, no. 2.

De Sanctis, M., Muccini, H., Vaidhyanathan, K. (2020). Data-driven Adaptation in Microservice-based IoT

Architectures. *In ICSA-C 2020, IEEE International Conference on Software Architecture Companion*. IEEE, Salvador, Brazil.

Gao, R. et al. (2021). Research on a Service-oriented Dynamic Adaptive Software Architecture. *In ICCSE 2021, 16th International Conf. on Computer Science & Education*. IEEE, Lancaster, United Kingdom.

Gomaa, H., Hashimoto, K., Kim, M., Malek, Menascé, D. A. (2010). Software Adaptation Patterns for Service-Oriented Architectures. *Proceedings of ACM Symposium on Applied Computing (SAC)*. Sierre, Switzerland.

Gomaa, H. (2011). *Software Modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press.

Gomaa, H., Hashimoto, K. (2012). Dynamic Self-Adaptation for Distributed Service-Oriented Transactions. *Proc. ACM/IEEE 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Zurich, Switzerland.

Gottschalk, S., Yigitbas, E., Engels, G. (2022). Model-driven Continuous Experimentation on Component-based Software Architectures. *In ICSA-C 2022, IEEE 19th International Conference on Software Architecture Companion*. IEEE, Honolulu, HI, USA.

Huynh, N.-T. (2019). State Transfer Management in Adaptive Software: An Approach from Design to Runtime. *In RIVF 2019, IEEE-RIVF International Conference on Computing and Communication Technologies*. IEEE, Danang, Vietnam.

Kaya, M.C., Eroglu, A., Karamanlioglu, A., Onur, E., Tekinerdogan, B., Dogru, A.H. (2019). Runtime Adaptability of Ambient Intelligence Systems Based on Component-Oriented Approach. *In Mahmood, Z. (Ed.), Guide to Ambient Intelligence in the IoT Environment, Computer Communications and Networks*. Springer, Cham.

Kramer, J., Magee, J. (1990). The evolving philosophers' problem: dynamic change management. *In IEEE Transactions on Software Engineering,* vol. 16, no. 11. pp. 1293-1306. Nov. 1990.

Kramer, J., Magee, J. (2007). Self-Managed Systems: an Architectural Challenge. *In FOSE '07, Future of Software Engineering*. IEEE, Minneapolis, MN, USA.

Mayrhofer, M., Mayr-Dorn, C., Zoitl, A., Guiza, O., Weichhart, G., Egyed, A. (2019). Assessing Adaptability of Software Architectures for Cyber-Physical Production Systems. *In ECSA 2019, Lecture Notes in Computer Science*, vol. 11681. Springer, Cham.

Mutanu, L., Kotonya, G. (2019). State of Runtime Adaptation in Service-Oriented Systems: What, Where, when, How and Right. *In IET Software*, vol. 13, pp.14-24.

Philip, M.M., Natarajan, K., Ramanathan, A., Balakrishnan, V. (2020). Composite Pattern to Handle Variation Points in Software Architectural Design of Evolving Application Systems. *In IET Software*, vol. 14, pp. 98-105.

Porter, J., Albassam, E. (2020). A Decentralized Approach to Architecture-Based Self-Protecting Software Systems. *In CCWC 2020, 10th Annual Computing and Communication Workshop and Conf.* IEEE, Las Vegas, NV, USA.

Shin, M. E., Gomaa, H., Pathirage, D., Baker, C., Malhotra, B. (2016). Design of Secure Software Architectures with Secure Connectors. *In International Journal of Software Engineering and Knowledge Engineering,* vol. 26, no. 05. pp. 769-805.

Shin, M., Gomaa, H., Pathirage, D. (2016). Reusable Secure Connectors for Secure Software Architecture. *In International Conference on Software Reuse. Springer*, Limassol, Cyprus.

Shin, M., Gomaa, H., Pathirage, D. (2017). Model-based Design of Reusable Secure Connectors. *In ModComp 2017, 4th International Workshop on Interplay of Model-Driven and Component-Based Software Engineering*. Austin, USA.

Shin, M., Gomaa, H., Pathirage, D. (2018). A Software Product Line Approach for Feature Modeling and Design of Secure Connectors. *In ICSOFT 2018, 13th International Conference on Software Technologie*s. SCITEPRESS, Porto, Portugal.

Shin, M., Gomaa, H., Pathirage, D. (2019). A Software Product Line Approach to Design Secure Connectors in Component-Based Software Architectures. *In CCIS, Volume 1077, Communications in Computer and Information Science.*

Shin, M., Kang, T., Gomaa, H. (2021). Design of Secure Connectors for Complex Message Communications in Software Architecture. *In ESSE '21, Proceedings of the 2021 European Symposium on Software Engineering.* ACM, pp. 21–28.

Shin, S.Y., Nejati, S., Sabetzadeh, M., Briand, L.C., Arora, C., Zimmer, F. (2020). Dynamic Adaptation of Software-Defined Networks for IoT Systems: A Search-Based Approach. *In SEAMS '20, Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, New York, NY, USA, pp. 137-148.

Vogel, T. (2018). MRUBiS: An Exemplar for Model-Based Architectural Self-Healing and Self-Optimization*. In SEAMS '18, Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing System*s. ACM, New York, NY, USA, pp. 101–107.