Integrating Security Protocols in Scenario-based Requirements Specifications

Thorsten Koch[1], Sascha Trippel[2], Stefan Dziwok[1] and Eric Bodden[3]
[1]*Fraunhofer IEM, Germany*
[2]*Paderborn University, Germany*
[3]*Paderborn University & Fraunhofer IEM, Germany*

Keywords: Scenario-based Requirements Engineering, Security Protocols, Simulative Validation.

Abstract: Software-intensive systems such as internet services, factories, or vehicles are characterized by complex functionality and strong interconnection. This interconnection leads to a high risk of cyber-attacks. To reduce this risk, software-intensive systems must fulfill various security requirements and integrate security mechanisms such as security protocols. Security protocols ensure secure communication between and within software-intensive systems. However, the application of security protocols could negatively impact other parts of the systems (e.g., its communication behavior) since the protocols introduce further messages and computing-intensive operations to the system's behavior. Therefore, the development of software-intensive systems needs to cover functional and security aspects. This paper presents a model- and scenario-based requirements engineering approach to integrate security protocols in application-specific requirements specifications systematically. Thereby, requirements engineers with limited security knowledge can integrate established and validated security protocols in their application to increase the security. In particular, our approach provides parameterizable templates for security protocols and references these templates in other specifications. Furthermore, it provides the simulative validation of the requirements specification. We show that our approach is applicable in practice through a case study involving application scenarios from the automotive domain and established security protocols.

1 INTRODUCTION

Software-intensive systems have become prevalent in our daily lives and are characterized by complex functionality and strong interconnection. However, the widespread use of software-intensive systems increases the risk of cyber-attacks significantly. Thus, security has become one of the most critical risks for the world's population (World Economic Forum, 2021) and one of the grand challenges in the field of model-driven engineering (Bucchiarone et al., 2020).

Attacks on technical systems like industrial control systems or automotive systems pose a high risk since malfunctions caused by security incidents can lead to life-threatening accidents. For example, (Miller and Valasek, 2015) conducted an attack on the Jeep Cherokee and were able to remotely control the vehicle and manipulate the brakes and the motor control.

To cope with the complex requirements on functionality and security, the development of software-intensive systems requires rigorous requirement engineering as detecting and fixing defects in subsequent development phases causes costly iterations.

Scenario-based approaches enable requirements engineers to specify what the system under development may, must, or must not do during its execution (Harel, 2001). The resulting requirements specification provides an intuitive representation of the system's behavior (Hassine et al., 2010) and is easy to understand for people with modeling experience (Abrahão et al., 2013). However, although security is a significant concern in developing software-intensive systems, scenario-based approaches currently only address functional and safety requirements. In addition, requirements engineers usually have limited security knowledge (Dziwok et al., 2021), which can lead to severe vulnerabilities in the system.

We presented a formal, model- and scenario-based requirements engineering approach for the specification and analysis of requirements on the

Koch, T., Trippel, S., Dziwok, S. and Bodden, E.
Integrating Security Protocols in Scenario-based Requirements Specifications.
DOI: 10.5220/0010783300003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 15-25
ISBN: 978-989-758-550-0; ISSN: 2184-4348

message-based communication behavior of software-intensive systems using Modal Sequence Diagrams (MSDs) (Holtmann et al., 2016).

Based on MSDs, we developed the *Security Modeling Profile* enabling the specification of security protocols in a scenario-based manner (Koch et al., 2020). Security protocols are used to secure the communication between two or more communication partners (Schneier, 2015). However, security protocols must be designed, implemented, and integrated correctly in the application. Using the Security Modeling Profile, security engineers can specify security protocols and verify whether the security protocol fulfills the desired security requirements by means of a security model checker. However, the specified and verified security protocols were not integrated yet into a requirements specification for an arbitrary application; instead, requirements engineers have to model the security protocols from scratch for each application.

To address the problem, this paper presents an approach to systematically integrate existing security protocols specified by means of the Security Modeling Profile in a scenario-based requirements specification. Therefore, we reduce the manual effort for a requirements engineer to use established and validated security protocols in his application. Furthermore, requirements engineers with limited security knowledge can use security protocols to increase the security of their application. In particular, this approach encompasses the specification of parameterizable templates for security protocols and referencing templates in the scenario-based functional requirements specification. Furthermore, we extend the play-out algorithm (Harel and Marelly, 2003), a simulative validation technique for MSDs, to validate whether the introduction of a security protocol causes any defects in the other requirements.

To evaluate our approach, we conduct a case study using two application scenarios from the automotive domain. Furthermore, we use five different security protocols from SPORE (the security protocols open repository) (Clark and Jacob, 2002)—among others—the Needham-Schroeder Public-Key protocol. Within our case study, we show that our approach is applicable and useful in practice.

The remainder of the paper is structured as follows. In the next section, we introduce the fundamentals of this paper. Section 3 presents our approach for reusable security protocols. In Section 4, we conduct a case study to evaluate our approach. Then, Section 5 covers related work. Finally, Section 6 concludes this paper with a summary and an outlook on future work.

2 MODAL SEQUENCE DIAGRAMS (MSDs)

This section presents the basic concepts of Modal Sequence Diagrams (MSDs) (Holtmann et al., 2016). The presented concepts have been implemented in the SCENARIOTOOLSMSD tool suite[1]. In Section 2.1, we introduce the overall structure of an MSD specification. Afterward, Section 2.2 presents the basics of the MSD semantics. Then, Section 2.3 describes the Play-out algorithm, an automatic validation technique for MSDs. Finally, Section 2.4 explains the *Security Modeling Profile* as an extension to MSDs that enables the specification and analysis of security properties on the communication behavior.

2.1 Structure of MSD Specifications

An MSD specification is structured by means of MSD use cases. Each use case encapsulates requirements on the communication behavior to be provided by the system under development. Therefore, an MSD use case encompasses the participants involved in a self-contained situation and a set of MSDs specifying the requirements on the communication behavior.

To illustrate the concepts of MSDs, we use the EBEAS (Holtmann et al., 2016), an advanced driver assistance system from the automotive domain, as a running example. EBEAS is supposed to reduce the risk of rear-end collisions. Figure 1 depicts a self-contained situation in which the vehicle leading detects an obstacle in front and has to perform an emerging brake. To avoid a rear-end collision, leading informs the vehicle ego about an emergency brake.

In an MSD specification, *UML classes* provide reusable types for all MSD use cases. These types are used to define the structure of the system under development and its environment. Environment objects are annotated with the stereotype *«Environment»* (e.g., Environment in Figure 1). Moreover, the UML classes define operations used as message signatures and define which messages can be received by a participant in an actual MSD. For example, the class diagram in Figure 1 depicts the two classes Environment and Vehicle. The class Environment encompasses the operation emcyBraking.

A UML collaboration (dashed ellipse in Figure 1) specifies the roles participating in a particular use case. The roles are typed by the UML classes and used as lifelines in an MSD. For example, the collaboration in the middle of Figure 1 encompasses the role ego that has an abstract syntax link type to the class Vehicle.

[1] http://scenariotools.org/projects2/

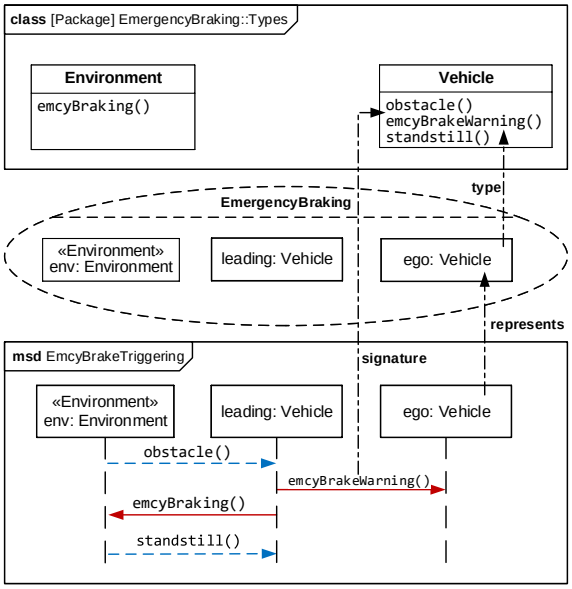

Figure 1: MSD Specification Excerpt for the Scenario Emergency Braking of the EBEAS.

Based on the UML classes and the collaboration, a set of MSDs specifies the requirements on the communication behavior between the roles involved in the MSD use case. For example, the bottom diagram in Figure 1 depicts such an MSD. An MSD encompasses lifelines and MSD messages. These messages are associated with a sending and a receiving lifeline and an operation signature. For example, the MSD message emcyBrakeWarning is associated with the equally named operation signature of the class Vehicle by means of the abstract syntax link signature. MSD messages sent from environment objects are called environment messages, whereas MSD messages sent from system objects are called system messages.

2.2 MSD Semantics

An MSD progresses as message events occur in a system at runtime or in an object system during the simulative validation by means of the play-out algorithm (Harel and Marelly, 2003).

A message event is unified with an MSD message if the event name equals the message name and if the sending and receiving lifelines of the message are bound to the sending and receiving objects of the message event.

When a message event occurs that is unifiable with the first message in an MSD, an active MSD is created. The active MSD progresses when other message events occur that are unifiable with the subsequent MSD messages. This progress is captured by the cut, which marks the locations of the passed MSD messages for every lifeline. If the cut is in front of an MSD message on its sending and receiving lifelines, the MSD message is enabled. If the cut reaches the end of an active MSD, the active MSD is terminated.

Each MSD message has a temperature and an execution kind. The temperature of a message can be hot or cold. The temperature is used to distinguish between provisional (cold) and mandatory (hot) behavior depicted by blue and red arrows in Figure 1. The semantics of a hot message is that other messages specified by the MSD are not allowed to occur. For example, the MSD messages emcyBrakeWarning and emcyBraking in Figure 1 are hot, whereas the others are cold. The execution kind of a message can either be executed or monitored depicted by solid and dashed arrows in Figure 1. An executed message indicates that the message must eventually occur, whereas a monitored message can but does not need to occur. For example, the MSD message obstacle and standstill are monitored, whereas the others are executed.

A violation occurs in an MSD if a message event occurs that is unifiable with a specified message but not enabled in this MSD. If a violation occurs, the active MSD is terminated. Based on the execution kind and temperature of a cut, we can define different types of violations.

- If a violation occurs in a cold cut (i.e., all enabled messages are cold), it is a cold violation, which does not violate the requirements.
- If a violation occurs in a hot cut (i.e., at least one enabled message is hot), it is a hot violation, which violates the requirements.
- If a violation occurs in an executed cut (i.e., at least one enabled message is executed), it is a liveness violation, violating the requirements.

2.3 Play-out Algorithm

The play-out algorithm (Harel and Marelly, 2003) enables the validation of an MSD specification. While simulating selected execution paths of the system under development, the play-out algorithm finds defects regarding the consistency and correctness of the requirements specification.

In the beginning, the system waits for environment events to occur. If an environment event occurs, MSDs are activated whose initial event is unifiable with the environment event. Next, the play-out algorithm chooses non-deterministically one of the enabled system events and executes it if this does not lead to a hot violation in another MSD. The process is repeated until there are no active MSDs. The system then waits for the next environment event. In case of a hot violation, the algorithm terminates.

2.4 Security Modeling Profile

The *Security Modeling Profile* (Koch et al., 2020) introduces an extension to the MSD profile to enable the specification of security requirements on the communication behavior. Therefore, the profile provides stereotypes to specify various security primitives (e.g., encryption or digital signatures).

Figure 2 depicts an excerpt of the MSD specification for the Needham-Schroeder Public-Key protocol (Lowe, 1996). The security protocol enables mutual authentication between two participants. For this purpose, the security protocol relies on a trusted key server, which stores and distributes the public keys of all participants.

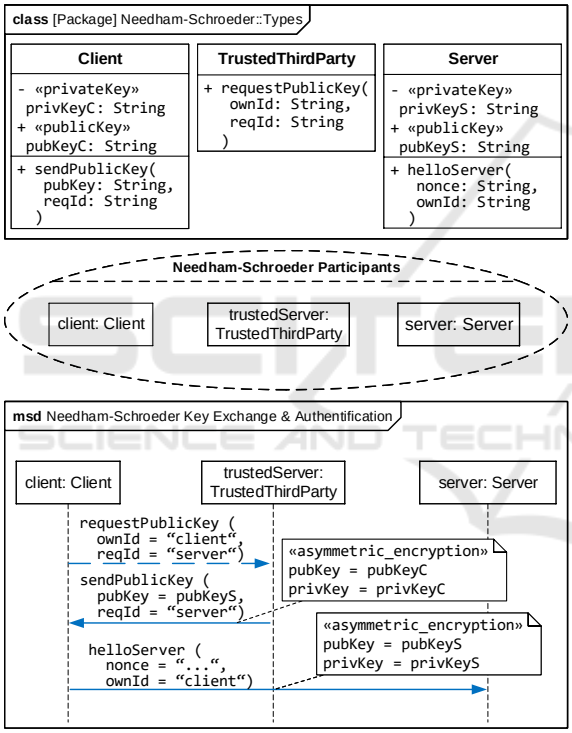

Figure 2: MSD Specification Excerpt for the Needham-Schroeder Public-Key protocol (using the Security Modeling Profile).

The MSD for the Needham-Schroeder Public-Key protocol, depicted at the bottom of Figure 2, contains the three lifelines client: Client, trustedServer: TrustedThirdParty, and server: Server. The client: Client sends the message requestPublicKey to the trustedServer: TrustedThirdParty to obtain the public key of the server: Server. The trustedServer: TrustedThirdParty replies to the request and sends the corresponding public key to client: Client (cf. sendPublicKey in Figure 2). According to the protocol specification, this message must be asymmetrically encrypted. Therefore, we applied the stereotype «asymmetric_encryption» to the message sendPublicKey. The stereotype has the two properties pubKey and privKey, which are necessary to express information for the encryption and decryption of the message. In the example depicted in Figure 2, the property pubKey is set to pubKeyC and the property privKey is set to privKeyC of the class Client, respectively.

To enable the verification of security protocols specified by the Security Modeling Profile, we conceived a model transformation from the enriched MSD specification into the security model checker ProVerif (Blanchet, 2001). In addition, the transformation generates all relevant queries that the model checker shall verify to decide whether the protocol is secure w.r.t. confidentiality and authentication.

Furthermore, they extended the MSD semantics such that the play-out algorithm can validate MSD specifications using the Security Modeling Profile. In particular, they extend the definition of message unification (cf. Section 2.3) and validate whether the same cryptographic primitives (e.g., stereotypes of the Security Modeling Profile) are applied to the message event and the corresponding message. Furthermore, the properties of the stereotypes of the message and the message event must be the same. For example, a message event that is unifiable with an enabled message but does not have the same cryptographic primitives leads to a violation.

3 INTEGRATION APPROACH

This section introduces our approach for integrating security protocols into scenario-based requirements specifications. First, we present our modeling approach to specify reusable security protocols and their application in scenario-based requirements specifications. Second, we describe the adaptation of the play-out algorithm to enable the simulative validation of the resulting scenario-based requirements specifications.

3.1 Modeling Approach

Security protocols can be used in different applications to ensure secure communication between the participants involved in the application.

To avoid the redundant and error-prone task of modeling the security protocol during the requirements engineering for a particular application, we propose that the security protocol specification is separated from the scenario-based requirements specification. Moreover, the scenario-based requirements

specification should reference the security protocol as depicted in Figure 3.

On a technical level, although the scenario-based requirements specification and the security protocol are specified by means of MSDs, a scenario-based requirements specification cannot reuse the behavior of the security protocol since both specifications rely on different types that are used as roles in the MSD use case and define the message that can be exchanged between the roles.

Thus, to overcome this problem, the types of the system under development must be related to the types of the security protocol. Thereby, the security protocol is adapted to the application context of the system under development, and the specified types can adopt the behavior of the security protocol types.

In Figure 3, we illustrate the initial situation using our running example. The application scenario Emergency Braking of the EBEAS, depicted in the MSD specification on the left side, is supposed to ensure the encrypted communication between the two roles leading: Vehicle and ego: Vehicle. Therefore, the requirements engineer decides to execute the Needham-Schroeder Public-Key protocol, depicted on the right side. After the execution of the protocol, the two roles know the public key of each other and can use it to secure their communication (cf. emcyBrakeWarning in Figure 3).

To adapt the security protocol to the application context, the role leading: Vehicle has to substitute the role client: Client; and the role ego: Vehicle has to substitute the role server: Server (cf. arrows labeled with represents in Figure 3). Furthermore, the properties that are used in the stereotypes to annotate that a message is encrypted (e.g., in Figure 3) must also be substituted by a property that exists in the scenario-based requirements specification.

One possible idea to relate the different classes is the concept of UML Inheritance. Therefore, the UML classes used as types of the scenario-based requirements specification extend the UML classes of the security protocols. The scenario-based requirements specification types inherit the operations and properties, which enable their representative roles to participate in the security protocol.

However, this solution has several drawbacks. First, as depicted in Figure 3, multiple roles in a scenario-based requirements specification can be typed by the same class. Thus, it would be unclear which role of the scenario-based requirements specification takes which role in the security protocol. For example, in Figure 3, the two roles leading and ego are both typed by the class Vehicle. Second, if a role of the scenario-based requirements specification executes the same security protocol with different participants, it might happen that the same properties of a type must be used in different contexts and, thus, would be overwritten. Therefore, it is necessary to define an explicit relationship for properties.

To address these drawbacks, we conceived a template-based modeling approach to adapt a security protocol to the application context of a scenario-based requirements specification.

In the following, we present the extension to the Security Modeling Profile to enable the specification of security protocol templates. Afterward, we introduce the approach to instantiate the templates within a scenario-based requirements specification.

3.1.1 Defining Security Protocol Templates

Our template-based modeling approach relates structural elements of the scenario-based requirements specification to structural elements of the security protocol. Thereby, the security protocol is adapted to the application context. As a consequence, we introduce the concept of template parameter. A template parameter is added to those elements of the security protocol that may be substituted by elements of the application. There are two types of template parameters: RoleTemplate and PropertyTemplate.

RoleTemplate: A security protocol encompasses at least two participants, the protocol initiator and the protocol responder. For example, in the security protocol depicted in Figure 3, the role client: Client initiates the protocol, and server: Server responds to the initiation. Furthermore, a security protocol can encompass other roles, e.g., a trusted third party as in the Needham-Schroeder-Key-Protocol.

To enable the substitution of the security protocol's roles, we use the stereotype RoleTemplate that can be applied to the roles of the UML collaboration. For example, in the security protocol depicted in Figure 3, we have applied this stereotype to the two roles client: Client and server: Server.

PropertyTemplate: In a security protocol, roles have prior knowledge that exists before the initial communication and is essential to the execution of the protocol. For example, in the security protocol depicted in Figure 3, the class Client typing the role client: Client has the two properties privKeyC and pubKeyC to represent the public/private key pair.

To enable the substitution of the security protocol's roles, we introduce a stereotype PropertyTemplate that can be applied to the properties

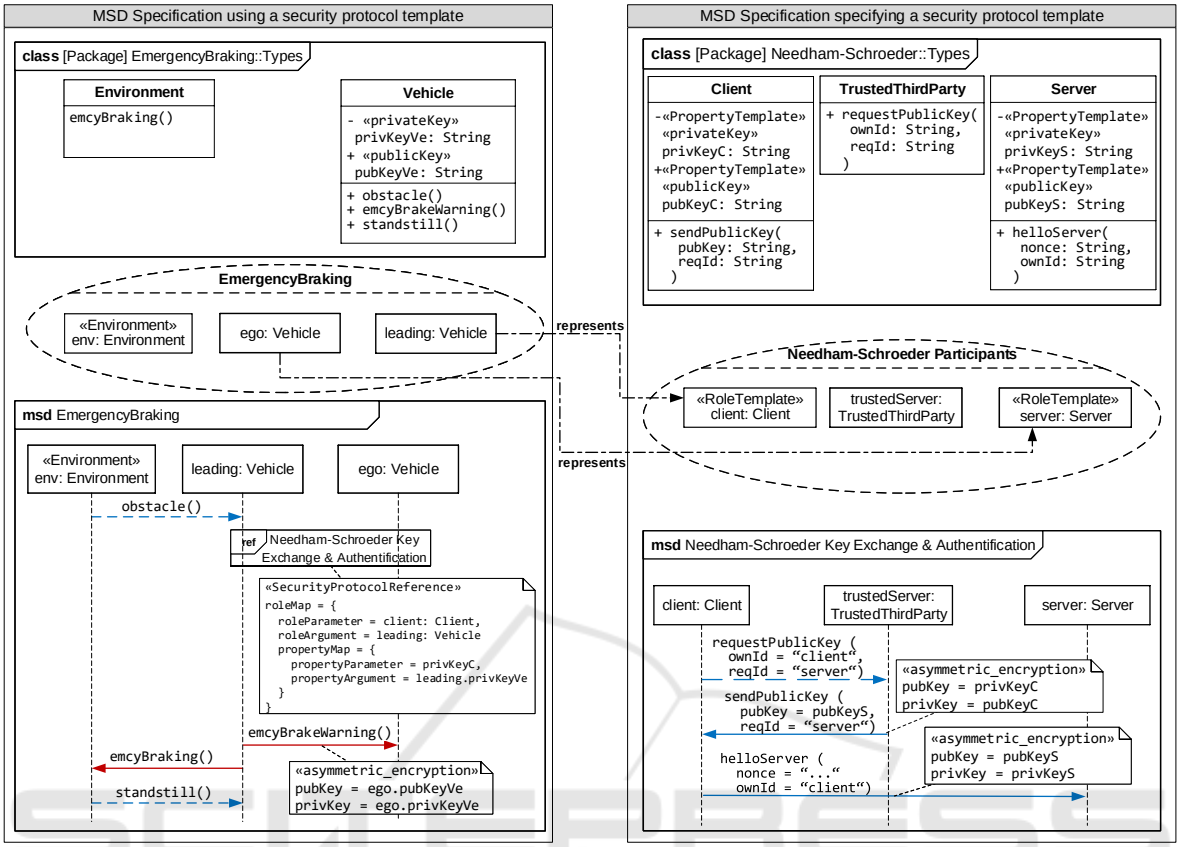

Figure 3: The Scenario-based Requirements Specification for the Application Scenario Emergency Braking is supposed to integrate the Needham-Schroeder Public-Key protocol specified in another specification. The SecurityProtocolReference in the MSD EmergencyBraking depicts only an excerpt of the role and property mapping.

of the UML classes. For example, in the security protocol depicted in Figure 3, we have applied the stereotype to all properties of the two classes Client and Server.

3.1.2 Referencing Security Protocol Templates

This section introduces our modeling approach for referencing security protocol templates based on UML InteractionUse (Object Management Group (OMG), 2017, Clause 17). A UML InteractionUse enables the reuse of existing interactions. However, a UML InteractionUse does not allow to specify structural substitutions.

Thus, we extend the Security Modeling Profile profile to specify the substitution of roles to roles and properties to properties while specifying the InteractionUse. Therefore, we create a new stereotype called *SecurityProtocolReference* that extends the UML InteractionUse.

The stereotype SecurityProtocolReference inherits the property refersTo of type Interaction. This property is used to specify the interaction that defines the behavior of the security protocol. For example, the SecurityProtocolReference in Figure 3 refers to the MSD Needham-Schroeder Key Exchange & Authentification.

In addition to the inherited properties, the stereotype SecurityProtocolReference encompasses a Role2RoleMap. The Role2RoleMap is a list of type Role2RoleMapEntry mapping the roles from the security protocol template to roles in the referencing MSD. The list has at least two elements, the initiator and the responder of the security protocol. The concrete mapping is specified in the data type Role2RoleMapEntry. Therefore, the data type has two properties: roleParameter and roleArgument. While the roleParameter corresponds to a role in the security protocol, the roleArgument captures a role of the application context.

Apart from the two properties used to define the role mapping, the Role2RoleMapEntry encompasses a property called Property2PropertyMap. It is used to map properties from the security protocol template to properties in the referencing security protocol.

For example, in the security protocol depicted in Figure 3, the stereotype SecurityProtocolReference specifies an excerpt of the role and property mapping. Here, the role client: Client of the security protocol is substituted by the role leading: Vehicle. Furthermore, the property privKeyC of the class Client is substituted by the property privKeyVe of the class Vehicle.

3.2 Simulation Approach

We adapted the play-out algorithm of ScenarioTools to support the modeling approach for the integration of security protocols in scenario-based requirements specification presented in the previous section.

The adaptation is restricted to the evaluation of the SecurityProtocolReference and the resolution of its defined properties. Apart from that, the algorithm behaves as described in Section 2.3.

If the cut of an active MSD is immediately before the SecurityProtocolReference, it is directly evaluated. Therefore, the play-out algorithm resolves the substitutions specified by the SecurityProtocolReference and adds the messages defined by the referenced security protocol. The substitution process encompasses two steps. First, the play-out algorithm evaluates the role mapping and substitutes the role accordingly. For example, Figure 4, depicts the MSD specification after resolving the SecurityProtocolReference. Here, the role leading: Vehicle substitutes the client: Client and the role ego: Vehicle substitutes the server: Server. Furthermore, the role trustedServer: TrustedThirdParty is added to the resulting specification since it is not part of any mapping. Note that it is not relevant whether the lifelines of the security protocol template are part of the environment or the system. Instead, the specification kind is taken from the referencing lifelines. As a result of the substitution, the roles of the requirements specification can now send and receive the messages of the security protocol. Afterward, the play-out algorithm evaluates the property mapping specified for each role mapping. All references to this property in the security protocol specification are searched for and replaced by the new reference. This applies to message parameters as well as to the references in the individual stereotypes of the Security Modeling Profile. After the two steps have been completed, the play-out continues as described in Section 2.3.

4 CASE STUDY

To evaluate the applicability and usefulness of our approach in practice, we conduct a case study based on the guidelines by (Kitchenham et al., 1995) and (Runeson, 2012).

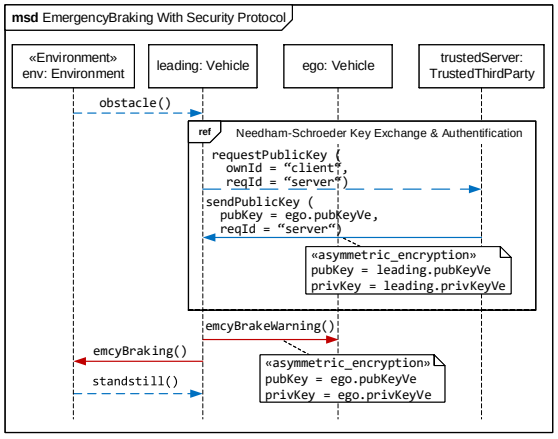

Figure 4: MSD Specification after the Template Substitution.

4.1 Case Study Context

We examine three evaluation questions (EQ):

EQ1: Does our approach enable the specification of security protocol templates for real-world security protocols?

EQ2: Does our approach enable the use of security protocol templates in a scenario-based requirements specification?

EQ3: Does our approach enable the analysis of scenario-based requirements specifications that include security protocol templates?

For this purpose, we use the Needham-Schroeder Public-Key protocol from our running example and selected four other security protocols from SPORE (the security protocol open repository) (Clark and Jacob, 2002). The five security protocols present a broad range of possible security protocols since they use different cryptographic primitives. For example, the Needham-Schroeder Public-Key protocol uses asymmetric encryption, while the Andrew Secure RPC uses symmetric encryption.

Furthermore, we model two application scenarios from the EBEAS (cf. Section 2.1) that shall integrate the five selected security protocols. The first application scenario is taken from the running example. The second application scenario is similar but contains two additional roles whose communication must be encrypted.

The two application scenarios are sufficient to evaluate our approach for the following two reasons. First, the use of the SecurityProtocolReference is independent of the application scenario. Here, it is

only essential to use different security protocols and, thereby, to specify various substitutions by means of the SecurityProtocolReference. Second, the simulation of a SecurityProtocolReference is also independent of the application scenario. Here, it is essential to use the same protocol multiple times but also to simulate different protocols at the same time. The second application scenario covers both cases.

In preparation of the case study, we implemented a prototype of our approach based on SCENARIOTOOLSMSD tool suite.

4.2 Research Hypotheses

Based on our objective and evaluation questions, we define the following three evaluation hypotheses:

H1: The five security protocols selected from SPORE can be specified as security protocol templates using our extensions to the Security Modeling Profile as presented in Section 3.1.1. We rate the hypothesis as fulfilled if each security protocol can be specified as a security protocol template using solely the approach presented in Section 3.1.1.

H2: All security protocol templates can be referenced in a scenario-based requirements specification. Therefore, all parameters specified by the template must be substituted by model elements of the scenario-based requirements specification using our approach presented in Section 3.1.2. We consider H2 as fulfilled if all security protocol templates can be used in the two scenario-based requirements specification correctly using our approach presented in Section 3.1.2.

H3: The references to the five security protocol templates can be evaluated correctly according to the runtime semantics defined in Section 3.2. We rate H3 as fulfilled if the contents of the security protocol template are correctly inserted into the referencing scenario-based requirements specifications during the simulative validation by means of the play-out algorithm.

4.3 Hypothesis Validation

To validate our hypotheses, we use the prototypical implementation of our approach based on the ScenarioTools MSD tool-suite. First, we model the MSD specification for each security protocol template and check whether our modeling profile is sufficiently expressive. Second, we use the two application scenarios as scenario-based requirements specifications and refer to the resulting security protocol templates. Finally, we validate the scenario-based requirements specifications, including the security protocol reference, through the play-out algorithm.

4.4 Analyzing the Results

The results of the case study are depicted in Table 1.

Using the extensions to the Security Modeling Profile, we were able to model all relevant information for specifying the five security protocols as security protocol templates. Thus, we consider H1 as fulfilled.

Furthermore, we specified the two exemplary scenario-based requirements specifications. To reference the different security protocol templates, we, first, added the necessary properties to the specification. Second, we added the SecurityProtocolReference and defined the substitution following the approach presented in Section 3.2. For both specifications, we were able to add new properties to the different types and, afterward, defined the substitution for all parameters of the referenced security protocol template. Thus, we consider H2 as fulfilled.

Finally, we executed the MSD specification that references the security protocol template using our adapted play-out algorithm. During the execution, the SecurityProtocolReference was evaluated. Subsequently, we observed that for each security protocol and simulation run the contents of the security protocol template were inserted correctly in the active MSD. Moreover, all messages were in the correct order and were sent between the lifelines specified as arguments to their original sending and receiving lifelines. Furthermore, the properties were correctly substituted and referenced where necessary. Thus, we consider H3 as fulfilled.

To conclude the case study, the fulfilled hypotheses indicate that our approach to specification, usage, and analysis of security protocol templates is applicable and useful in practice.

4.5 Threats to Validity

The following facts threaten the validity of the case study. First, we only modeled a selection of possible security protocols as security protocol templates. So, we cannot generalize our results for all possible security protocols. However, this threat is mitigated because the selected security protocols are typical examples of a security protocol covering all aspects relevant to security protocol templates. Second, the security protocol templates are only used by two exemplary scenario-based MSD specifications. This threat is mitigated because the evaluation of a security pro-

Table 1: Results of the Case Study.

Security Protocol	**H1** Modeling as Security Protocol Template	**H2** Integration in Application Scenarios	**H3** Validation of the Resulting Specification
Andrew Secure RPC	✓	✓	✓
Denning-Sacco Shared Key	✓	✓	✓
Diffie-Hellman Key Exchange	✓	✓	✓
Needham-Schroeder Public-Key	✓	✓	✓
Needham-Schroeder Symmetric-Key	✓	✓	✓

tocol template reference is independent of other parts of the MSD specification (cf. Section 3.1.2). Finally, the case study was performed by the same authors that developed the approach. Since the researchers might be biased toward the developed approach, the case study would be more significant if other security experts had modeled the security protocol templates and if requirements engineers had modeled the scenario-based specifications.

5 RELATED WORK

Some approaches consider the specification of security mechanisms and their reuse in an application's requirements and design phase.

(Mouheb et al., 2009) propose an aspect-oriented modeling approach to integrate security mechanisms (e.g., security protocols like TLS) into software design models (e.g., UML Interactions). Therefore, the authors present a UML profile to specify security mechanisms. This profile introduces a transparent and automatic approach that weaves the security mechanism into the design models. The approach is similar to ours since the authors present an approach to reuse security mechanisms that people with limited security knowledge can use. However, while the authors only consider the behavior of a security mechanism, we also consider structural properties. Thereby, we can better adapt the security mechanisms to the context of the application. Furthermore, we provide a simulative validation of the resulting specification.

(Ray et al., 2004) present an approach to incorporate role-based access control (RBAC) policies into the design of applications. The RBAC policies are defined independently of the application that has to implement the policies. To bridge the gap between the policy definition and the application design, Ray et al. specify policies by means of UML diagram templates. These UML diagram templates can be used to integrate application-specific policies into the design of the application. The approach is similar to our approach since the authors present an approach to reuse security mechanisms. Furthermore, both approaches enable the validation of the resulting specifications. However, in contrast to our approach, Ray et al. consider role-based access control and only consider the specification and analysis of structural models like UML class and object diagrams.

Furthermore, many approaches consider the modeling of system and software security using UML and SysML. For example, (Jürjens, 2002) propose UMLSec as an model-driven approach for integrating security-related information in UML specifications. UMLSec encompasses a UML profile for expressing security mechanisms, including secure information flow, confidentiality, and access control.

(Lodderstedt et al., 2002) present SecureUML, a UML-based modeling language for model-driven security. The approach enables the design and analysis of secure, distributed systems by adding mechanisms to model role-based access control. Furthermore, they provide an automatic generation of access control infrastructures based on the specified models.

(Roudier and Apvrille, 2015) present SysML-Sec, a modeling approach based on SysML to enable the specification of security aspects for embedded systems. They enhance SysML block and state machine diagrams to capture security-related features. In contrast to the other two approaches, SysML-Sec also covers safety-related features.

However, all these approaches mainly focus on extending the UML/SysML notations to reflect security concerns better. In contrast, our approach addresses the systematic reuse of existing security mechanisms. Thereby, our approach separates the specification and application of security mechanisms and allows people with limited knowledge to rely on these mechanisms to secure their applications.

6 CONCLUSION

This paper presents an approach for the systematic integration of security protocols templates in scenario-based requirements specifications. Therefore, we

contribute three building blocks. First, we extend the UML-based Security Modeling Profile with language constructs enabling the specification of security protocol templates. Second, we introduce an approach to refer to such a security protocol template in scenario-based requirements specification based on the UML-based scenario formalism MSD. Third, we extend the simulative validation technique play-out to support the security protocol templates.

Our approach enables requirements engineers to adapt existing security protocols to their context and, thus, integrate them systematically into their application to increase security. In addition, the integration makes the protocols accessible to people with little security knowledge. Finally, the simulative validation enables requirements engineers to check whether introducing security protocols has violated other requirements.

Future work encompasses three aspects. First, we plan to conduct a user study to evaluate whether our modeling language is intuitive and easy to use for security engineers and requirements engineers. Second, we want to collect typical security protocol templates in a library. Requirements engineers could search this library and select the protocol that best suits their needs. Third, we want to improve the security modeling abilities of our methodology. Therefore, we want to introduce the concept of misuse cases and make them analyzable by means of the play-out algorithm.

ACKNOWLEDGEMENTS

This research has been partly sponsored by the project "AppSecure.nrw - Security-by-Design of Java-based Applications" funded by the European Regional Development Fund (ERDF-0801379).

REFERENCES

Abrahão, S., Gravino, C., Insfran, E., Scanniello, G., and Tortora, G. (2013). Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments. *IEEE Transactions on Software Engineering*, 39(3):327–342.

Blanchet, B. (11-13 June 2001). An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001*, pages 82–96. IEEE.

Bucchiarone, A., Cabot, J., Paige, R. F., and Pierantonio, A. (2020). Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, 19(1):5–13.

Clark, J. and Jacob, J. (2002). Security protocols open repository.

Dziwok, S., Koch, T., Merschjohann, S., Budweg, B., and Leuer, S. (2021). AppSecure.nrw Software Security Study. https://arxiv.org/abs/2108.11752.

Harel, D. (2001). From play-in scenarios to code: an achievable dream. *Computer*, 34(5):53–60.

Harel, D. and Marelly, R. (2003). *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer.

Hassine, J., Rilling, J., and Dssouli, R. (2010). An evaluation of timed scenario notations. *Journal of Systems and Software*, 83(2):326–350.

Holtmann, J., Fockel, M., Koch, T., Schmelter, D., Brenner, C., Bernijazov, R., and Sander, M. (2016). The MechatronicUML Requirements Engineering Method: Process and Language. Technical Report tr-ri-16-351, Software Engineering Department, Fraunhofer IEM / Software Engineering Group, Heinz Nixdorf Institute.

Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. In Jézéquel, J.-M., editor, *The unified modeling language*, volume 2460 of *Lecture Notes in Computer Science*, pages 412–425. Springer, Berlin [u.a.].

Kitchenham, B., Pickard, L. M., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE Software*, 12(4):52–62.

Koch, T., Dziwok, S., Holtmann, J., and Bodden, E. (2020). Scenario-Based Specification of Security Protocols and Transformation to Security Model Checkers. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, page 343–353, New York, NY, USA. Association for Computing Machinery.

Lodderstedt, T., Basin, D., and Doser, J. (2002). SecureUML: A UML-Based Modeling Language for Model-Driven Security. In Jézéquel, J.-M., Hussmann, H., and Cook, S., editors, *≪UML≫ 2002 — The Unified Modeling Language*, pages 426–441, Berlin, Heidelberg. Springer Berlin Heidelberg.

Lowe, G. (1996). Breaking and fixing the needham-schroeder public-key protocol using fdr. In Goos, G., Hartmanis, J., Leeuwen, J., Margaria, T., and Steffen, B., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Berlin Heidelberg, Berlin, Heidelberg.

Miller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*.

Mouheb, D., Talhi, C., Lima, V., Debbabi, M., Wang, L., and Pourzandi, M. (2009). Weaving Security Aspects into UML 2.0 Design Models. In *Proceedings of the 13th Workshop on Aspect-Oriented Modeling*, AOM '09, page 7–12, New York, NY, USA. Association for Computing Machinery.

Object Management Group (OMG) (2017). OMG Unified Modeling Language (OMG UML) – Version 2.5.1. OMG Document Number: formal/2017-12-05.

Ray, I., Li, N., France, R., and Kim, D.-K. (2004). Using Uml to Visualize Role-Based Access Control Constraints. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*, SACMAT '04, page 115–124, New York, NY, USA. Association for Computing Machinery.

Roudier, Y. and Apvrille, L. (2015). SysML-Sec: A model driven approach for designing safe and secure systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 655–664.

Runeson, P., editor (2012). *Case study research in software engineering: Guidelines and examples*. Wiley, Hoboken, N.J, 1st ed. edition.

Schneier, B. (2015). *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons Incorporated, New York.

World Economic Forum (2021). *Global risks 2021: Insight report*. World Economic Forum, Geneva, 16th edition edition.