Building Secure Systems Using a Security Engineering Process and Security Building Blocks [*]

Andre Rein, Carsten Rudolph, Jose Fran. Ruiz

(andre.rein, carsten.rudolph, jose.ruiz.rodriguez)@sit.fraunhofer.de

Abstract:
In today's software development process, security related design decisions are rarely made early in the overall process. Even if security is considered early, this means that in most cases a more-or-less encompassing security requirements analysis is made. Based on this analysis best-practices, ad-hoc design decisions or individual expertise is used to integrate security during the development process or after weaknesses are found after the deployment. This paper explains the SecFutur security engineering process with a focus on Security Building Block Models which are used to build security related components, namely Security Building Blocks. These Security Building Blocks represent concrete security solutions and can be accessed via SecFutur patterns on the level of domain-specific models for particular application domains. The goal of this approach is to provide already defined and tested security related software components, which can be used early in the overall development process, to support security-design-decision already while modeling the software-system. Security Building Blocks are discussed in the context of the SecFutur Security Engineering Process with its requirement analysis and definition of security properties.

1 Introduction

Considering security in all phases of a system development process means to introduce an additional view in all phases including among others requirements specification, design decisions, implementation or documentation. Further, in many cases expert knowledge on security topics is essential for the development of secure systems. The SecFutur project [SF] develops a process that provides this additional view through UML-based security models of a system and combines these with support for design decisions based on security building blocks that are also modeled using UML.

Large parts of the SecFutur process use domain-specific artifacts. The domain-independent parts are a core security meta-model defining the concepts used in modeling security and the rather technical security building blocks describing how to use concrete implementations of security functions. Domain-specific parts include domain-specific security models, specific system models and security patterns that describe how security building blocks can be used and combined within a particular application domain to satisfy specific secu-

[*]This work was supported by the EU FP7 project SecFutur

rity requirements.

The Goal of SecFutur is to provide a modular modeling framework that allows the creation of precise and pluggable representations of the specialized knowledge of different application domains. The design of the representation mechanisms must also take into account the different roles involved. The specialized knowledge must serve different purposes. Of course, it should increase the security of the system. Further, documentation of security requirements and security design decisions is very important in the engineering process and should be tool-supported. Finally, the framework should provide useful knowledge to system engineers (security requirements, threats to consider, available solutions, trust models,. . .) and help developers to correctly implement and / or integrate security solutions, do optimization, testing, assurance, etc.

This paper describes the SecFutur security engineering process and focuses on the model for security building blocks (SBBs). These SBBs are a core element of the SecFutur process, as they represent domain-independent security functions and help developers to understand how to securely use particular security functions. Furthermore, in addition to interfaces, conditions and information on the functionality, SBBs also provide information on the set of assumptions that needs to be satisfies in the system. These assumptions can then either be realized by additional building blocks or can be subject to risk analysis and concrete threat analyses.

2 State of the Art

A large variety of security technology and individual security solutions exists that can be used by system developers to support security in their systems. Thus, in principle, a developer should be able to take security design decisions early in the development process and just choose from the large set of available security solutions this idealized view of development processes is only valid in very rare cases. The usual approach is a mixture of more-or-less systematic security requirements analysis, ad-hoc design decisions, some best practices, individual security expertise and finally step-by-step improvement after weaknesses have been found either by security testing or in the deployed product.

Another interesting aspect is that only a rather small set of available security solutions is actually used in real-life products and is obviously not available in the current design processes. One prominent example is the Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG) [Gro]. A very interesting work [Pea02] describes TPMs as the future elements for security. This security chip is already available in thousands of notebooks and laptops. A TPM potentially can provide various security functionalities that can be used to secure network connections, monitor the security of devices, securely store data, etc. However, only a very small subset of installed TPMs is actually activated and used. Furthermore, if it is used only a very small subset of available functions of the TPM is involved. Experience with developers interested in applying the TPM has shown that one of the reasons for the missing uptake of this technology is the complexity of the specifications. It is not clear how combinations of TPM commands (or TCG software stack commands) can actually implement some particular security services. Thus, making advanced security functionality available for development processes is a challenge.

A different approach for security solutions are security patterns. Usually, technical details and implementation details necessary for development are not included in the concept of security patterns [Ste06, Roe01]. Some interesting works are the one presented by H. Lohr et al. [LSW10] where he presents patterns for secure boot and secure storage in computer systems, the security patterns for mobile ad hoc networks developed by Jayraj Signh et al. [SSS11], security patterns for agent systems defined by Paolo Giorgini et al. [MGS03] or the work for architecting software with security patterns done by Riccardo Scandariato et al. [SYHJ08]. Although there exist more works for using security patterns as security solution of systems one problem they have is that they do not cover all the different phases of the creation of a system, as they are used for design or implementation but not in the modeling phase. This can create several problems, being one of them the outdating of the models of the system. For example, if, after modeling the system, an engineer uses a security pattern that needs additional elements such as a database or key-store the system will be modified with elements not defined in the modeling phase. These new elements can create new configurations or security flaws that were not expected. For that reason, our approach uses the idea of security patterns and extends it with Security Building Blocks (SBBs). They are represented by UML models in order to describe the functionality and characteristics of security properties in a real world scenario. These SBB models reflect security related software components, which are encapsulated abstractions of program functionalities. Software abstraction, encapsulation and information hiding build the basis of those SBBs. The main focus of using Building Blocks has always been reusability, maintainability and documentation. An interesting work [LSW87] describes these basic concepts with relation to General Building Blocks used in software development. Consequently, this work tries to refine those general concepts and apply them in the field of security, to model and build more secure systems.

The security patterns are not used directly with the security requirements of the system. They are connected to them by means of the security properties defined in the Domain Security Meta-models (explained in the following section). Each security property provides the solution as a security pattern and this one provides a set of SBBs for the implementation of the solution. Thus, the security patterns are not related to the security requirements of the system but to its security properties. This way the modeling phase and implementation phase of a system are related in a direct way and it is naturally embedded in the system.

3 Security Engineering Process

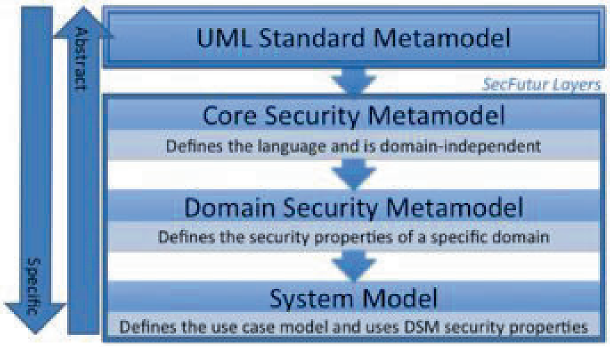
The development of systems composed of embedded components is a very complex task due to their specific characteristics and nature. Many systems of embedded components are composed of a lot of different embedded devices, such as the smart metering system. In these systems, many smart metering devices obtain the information of metering from many houses, process and send it to a different node. This node checks, processes, stores, etc. the information and sends it to another node, which works with the information provided from many nodes as that one. The systems of embedded components has a reactive nature too. When they process information they may need to react in a specific way, e.g. activating other systems, sending information, etc. One example is the forest control system, where, if they detect a fire, they have to send an alarm. Following this last example we can see that these systems have a real-time nature. They obtain the information, process it and work in real-time. For example, in the Mobile ad-hoc Network (MANET), the nodes enter and exit the system without warning, so the system must react in real-time to these changes and act accordingly. These systems use many components and resources, being hardware or software. For example, they can work with external components such as transmitters, video cameras, sensors or resources such as key-stores, databases, APIs, TPMs, etc.

Through some research done by the companies involved in the SecFutur project and others related to the development of TPM systems, real-time systems, etc. it was learnt that companies usually do not follow a clearly defined engineering process for the development of these systems. Their way of work is start developing and implementing as soon as they can. They sometimes follow a methodology but their focus is to start developing and adding functionalities as they find it necessary. This implies that security is either implemented later in the process as an extra feature or just ignored. Sometimes, when the functionality of the system is almost complete they start adding security. Obviously, in this stage, security is not naturally integrated in the system. Of course, this observations does not hold for systems with strong safety regulations, where strict waterfall development models are in place. In such a clearly structured development process, current threat-based approaches are also not suitable, because the detection of threats and high risks later in the development process requires to start-over and go back to the requirements phase.

SecFutur proposes a security engineering process that allows to develop and use security solutions in order to satisfy the security requirements of systems of embedded components. It integrates, in a flexible way, security solutions in a framework for the development of systems composed of embedded components. Its main objective is to help developers and engineers in the management of security aspects and its use in System Models. The process can be applied to existent processes, improving the security functionality of any process used to model a scenario. Due to size limitations it is not possible to explain all the details and characteristics of the SecFutur Security Engineering Process. A more complete description of the process itself can be found at [RHM11].

Some of the most important characteristics of the process are:

- It helps system developers in making design decisions for finding the best solution for their systems
- It facilitates the certification and the national / international regulations of the security artifacts
- It satisfies the specific requirements of the systems
- The implementation solutions are provided by means of SecFutur Patterns (SFPs) and Security Building Blocks (SBBs), where domain-specific SFPs define how domain -independent SBBs are used and composed within a particular domain.



UML Standard Metamodel
SecFutur Layers
Core Security Metamodel
Defines the language and is domain-independent
Domain Security Metamodel
Defines the security properties of a specific domain
System Model
Defines the use case model and uses DSM security properties
Abstract
Specific

Figure 1: SecFutur Layers

3.1 Artifacts

The different artifacts of the security engineering process have specific objectives and functionalities. Figure 1 describes the artifact structure.

The SecFutur layers go from more to less abstraction and specification. The upper one, the Core Security Meta-model (CSM), is the most abstract. It is based on the UML Standard Meta-model. This meta-model is used as basis for the definition of the different UML elements used in the creation of the CSM such as classes, relations, attributes, etc. The CSM defines the grammar and language for the definition of the domain-specific security artifacts. Because of that, the CSM is domain independent and only defines the abstract architecture. The specification of the security properties and characteristics of each domain is done in the Domain Security Meta-model (DSM). This one uses the CSM as basis because it defines the language. Finally, the System Model is the most specific layer. It is the model of the use case. In this model the system engineer imports a DSM (or various DSMs) and apply its security properties in order to fulfill the security requirements of the system. As we said before, due to the size limitations the reader can find more information of these layers in [RHM11].

The DSMs, as we explained before, define the specification of the domain security knowledge. It allows experts to capture their security knowledge (properties, solutions, threats, etc.) related to specific scopes (standards, company policies, etc.) in a specific domain (Mobile ad hoc Network, Smart meters, etc.). The security properties defined in a DSM are related to implementations by means of SecFutur Patterns (SFP) and Security Building Blocks (SBBs). The security properties define the characteristics and attributes of the solution and the SFP and SBBs their implementation using software / hardware elements. A SecFutur Pattern is a evolved version of the traditional security patterns [Ste06, Roe01], adapted and extended to the SecFutur Engineering Process. It provides information such as the security properties provided and the elements of the system where they can be applied, some examples of use (with support for computer-processing), the elements of the system model that must inter-operate with the pattern elements, a list of restrictions and metrics of the pattern with regards to the security properties defined before, the elements that must be added to the system in order for the pattern to work (this part is done by using the Security Building Block Models (SBBMs), a series of rules (in OCL format) used to verify the sound integration of the pattern in the system, a series of assumptions that apply to the system once the pattern has been integrated, some known uses and finally related patterns. Due to size limitations we only do a superficial description of this element. Thus, each security property is attached to a SFP, which defines its implementation by means of a SBBM and SBBs. The Security Building Block Models define the structure, relations and elements of the solution. Its basic elements are the SBBs. A SBB (or the aggregations of several SBBs) can provide the implementation solution for a specific property in a specific DSM. Resuming, each security property of a DSM has a SFP that describes its solution, which is implemented by means of SBBs. Following we describe these elements, its characteristics and functionality.

The creation of a DSM involves two different steps. First the analysis of the domain and second the definition of the different security properties. Although the two steps are out of scope of this paper we describe them briefly so the reader can understand better how the Security Building Block Models and the Security Building Blocks provide solutions to a great number of security properties of different domains. Briefly, the analysis of the domain checks the possible security threats and security properties of the system. This analysis provides the necessary information for the definition of the security properties. Once the security domain expert has the information of the domain, she starts modeling security properties. Each security property is composed of several elements such as its threats, assumptions, certifications, V&V (Validation & Verification) elements, etc. After the security domain expert defines a security property she searches for the SBB model that can provide a solution. The search of the solution is done by checking the characteristics of the security property in the list of SBB models. The SBB models are defined by the security property they fulfill, some requirements of the system (such as the elements they need in order to work correctly), the domain, etc. When the SBB model is found, it is attached to the SFP and then linked to the security property. If a combination of SBBs is needed, a more complex SFP needs to be build and validated / verified within the context of the DSM.

4 Security Building Blocks

In contrast to a security pattern, one single Security Building Block does not describe a complex integrated security solution. SBBs should be seen as encapsulated components that are domain-independent and can interact with other components in order to provide a clearly defined security service. The concept of abstracting software functionalities in SBBs can use other SBBs and can also interact with other components in a clearly defined way. In principle, a SBB can be just a concrete implementation of a security solution. However, in order to integrate a SBB into the engineering process, a description of the SBB is required. Here, this description is done in terms of a UML model. Thus, a so-called SBB Model represents one (or several) instantiations (i.e. implementations) of the

SBB.

As SBBs may reflect concrete implementations of a Security Solution, they also need to provide an interface which defines method names and data types used by the SBB. On one hand this is documentation for the system modeler, to better understand how the SBB can be integrated in the system model. On the other hand it serves as a concrete specification how the SBB may be implemented in a concrete realization. In addition to the security properties (or security service) provided by the SBB, this model also needs to provide information on preconditions and constraints, as well as on postconditions on the system that need to be fulfilled after the SBB was applied.

The SBB Meta-model, which is described in detail in Section 4, defines all the different artifacts and their relationships which were concisely presented in this section.

4.1 The Security Building Block Meta-model

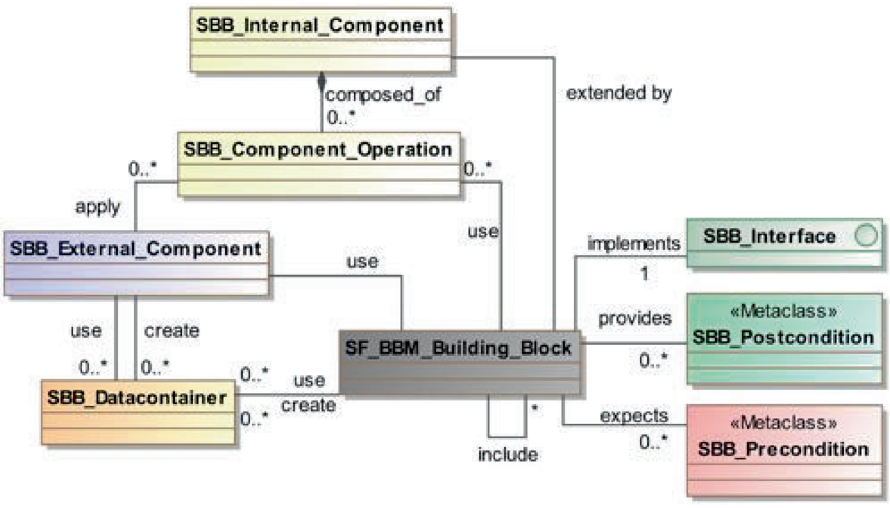


Figure 2: SBB Meta-model

The Security Building Block Meta-model (SBB Meta-model), as shown in Figure 2, delineates artifacts and relationships to construct Security Building Block Models (SBBM). More concretely, the SBB Meta-model acts as a determining factor to depict one or more Security Building Blocks and their interactions with other artifacts to build a SBBM.

Different artifacts enable the SBBM designer to describe their SBBMs in a concrete way (as a detailed view of internal SBBM components), but also leave the possibility to describe interfaces which may be used from a system model. Although a concrete SBBM and its SBBs may be used in concrete implementations, its main purpose is helping to represent Security Properties at the implementation level. These implementations provide the solutions of the security properties defined in the Domain Security Meta-model. The solutions are specified by means of Security Patterns.

Each SBBM and so their SBBs comes with its own dependencies and conditions. These dependencies may be whole system components (databases, key-chains, TPMs or other external components), simple data structures (cryptographic keys, plain-text / encrypted data, etc.) or even other SBBs. In a later stage of the development of the SecFutur tools, all these dependencies should be resolved automatically and imported into the concrete system model after a Security Pattern is selected as a solution for a specific Security Property.

The following presents a more detailed explanation of the artifacts shown in Figure 2.

4.2 Artifacts and Interactions

4.2.1 SBB_Datacontainer

The SBB_Datacontainer artifact is the most general type in the SBB Meta-model. It is used as a container for any kind of data which needs to be processed by a SBB. It may appear in the model as an output value of an external component (*SBB_External_Component creates SBB_Datacontainer*) or of a SBB (*SF_BBM_Building_Block creates SBB_Datacontainer*). Additionally a SBB may use it as a input value (*SF_BBM_Building_Block uses SBB_Datacontainer*).

4.2.2 SF_BBM_Building_Block

The SF_BBM_Building_Block artifact (SBB) is the key components of any SBBM. SBBs are used to represent any kind of security-related system components and encapsulate them in a single artifact which is used in the SBBM. A SBB may be defined broadly in an early stage of the SBBM and refined more detailed as soon as more information is needed or provided. Since a SBB may be composed of other SBBs the level of detail may be increased during the SBBM development process when it is needed or required. On the other hand it is also common that SBBs are used to compose a more complex SBB. This is also independent from the level of detail of any single involved SBB and depends only of the desired level of abstraction of the SBBM. Any artifact from the SBB Meta-model has at least one direct relationship to a SBB.

4.2.3 SBB_Precondition

A *SBB_Precondition* is an requirement which specifies under what conditions a SBB may be applied successfully (*SBB expects SBB_Precondition*). A single SBB_Precondition represents exactly one requirement, which may be formal or informal. It is designated that for any different requirement a single artifact instance is used. For example if an input value of a SBB is a cryptographic key the SBB_Precondition may determine that its size must be at least 128Bit. Additionally another SBB_Precondition may determine that a random number for the key generation may only come from a source considered secure (e.g. ”/dev/random” instead of ”/dev/urandom” in Unix Systems). Preconditions can be either satisfied by other SBBs (to be defined in a SecFutur Pattern) or remain as assumptions for the final system that need then to be evaluated for the environment the system should run in. This evaluation of remaining assumptions will be part of a risk analysis. In security certifications, these assumptions express policies for the operational environment.

4.2.4 SBB_Postcondition

A *SBB_Postcondition* is a statement that is valid if a SBB is applied successfully (*SBB provides SBB_Postcondition*). A successful application implies that any SBB_Precondition was obeyed. For example a SBB which encrypts confidential data under a given key may assert that the output data may be protected against eavesdropping. If a SBB has multiple assertions which become valid after a successful application, each different statement must appear as a single artifact instance. Again a SBB_Postcondition may be formal or informal.

4.2.5 SBB_External_Component

A *SBB_External_Component* is a system component which must be available for a SBB to function properly, but lies out of scope of the current SBB or even the SBBM. A SBB may use the functionality of the external component either by using its functionality directly (*SF_BBM_Building_Block uses SBB_External_Component*) or by using data structures that are produced by it (*SF_BBM_Building_Block uses SBB_Datacontainer created_by SBB_External_Component*) . If a Security Pattern is selected which involves external components the system engineer is informed about what specific external components are needed. Either the system engineer must provide these components from within their own system model or they are created automatically represented by additional interfaces or even concrete implementations. Another possibility is that a SBB_Datacontainer is used by a SBB_External_Component as an input value *(SBB_External_Component uses SBB_Datacontainer*).

Additionally an SBB_External_Component may apply functionalities of an SBB_Internal_Component, by using its SBB_Component_Operations (*SBB_External_Component applies SBB_Component_Operation*). For example using a SBB implementation which involves a TPM, many functionalities are based around the reporting of a system state. To keep track of the system state, an external component, namely IMA (Integrity Measurement Architecture) is used. IMA applies an operation of the TPM, which modifies internal registers in the TPM. These registers, which reflect the current system state, are later used in SBBs which need this system state for their own functionality. In consequence it is mandatory to distinguish between external and internal components. Both components must be available, but an external component represents a part of the system where the SBBM or a concrete SBB has no direct influence.

4.2.6 SBB_Internal_Component

The *SBB_Internal_Component* represents a part of the SBBM which is directly associated with a SBB over its SBB_Components_Operations. Any internal component consists of SBB_Component_Operations, which represent a concrete functionality of the component. A SF_BBM_Building_Block may modify a SBB_Internal_Component (*SF_BBM_Building_Block modifies SBB_Internal_Component*), which is considered as an unspecified usage of a SBB_Component_Operation within the SBB.

Additionally a SBB may modify a internal component (*SBB_Internal_Component extended _by SF_BBM_Building_Block*) such that it enhances the functionality of the component. This operation is also not precise and should be explained in detail depending on the enhancement (e.g. with additional diagrams or textual).

4.2.7 SBB_Component_Operation

A *SBB_Component_Operation* is an operation of an internal component which may be executed by a SBB (*SF_BBM_Building_Block uses SBB_Component_Operation*). This usage is handled internally in a SBB and is mostly a call of a function or method, provided by a library of the internal component, which executes the components real operation.

4.2.8 SBB_Interface

The *SBB_Interface* describes the public interface which may be used by a system engineer or other SBBs which need to integrate the current SBB. The application of the SBB is limited to just this specified operations and thus the only way to communicate with the SBB. The SBB_Interface serves as documentation for the input and output values as well as the description of the functionality. This information is mandatory for a system engineer who wants to use SBB in a concrete system model. Additionally the SBB_Interface is used when SBBs are combined with each other. This aspect is described in more detail in Section 5.3 and 5.4.

5 Example Model

Assuming there exists a SBB which simply encrypts data by using a symmetric cipher[1], as shown in Figure 3. This SBB needs at least data which should be encrypted (Data) and, additionally, a key (Key) that is used to apply the encryption. After the data is encrypted by the SBB, an output value is created which contains the original data encrypted under the given key (EncryptedData). A system engineer, who wants to integrate SBB_Encryption in a system model, needs the information about all input and output values of a SBB.

[1]For the used example a concrete encryption algorithm was intentionally left out. An encryption SBB which is used in a real world scenario must always specify a concrete algorithm (like AES-128 in CBC mode) or provide a mechanism to select / propose a proper algorithm.

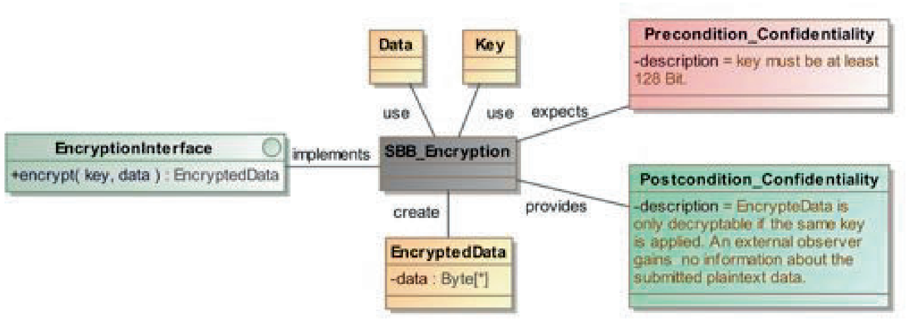

Data
Key
Precondition_Confidentiality
-description = key must be at least 128 Bit.
use
use
expects
EncryptionInterface
+encrypt(key, data) : EncryptedData
implements
SBB_Encryption
create
provides
Postcondition_Confidentiality
-description = EncrypteData is only decryptable if the same key is applied. An external observer gains no information about the submitted plaintext data.
EncryptedData
-data : Byte[*]

Figure 3: Combining SBBs

Although the SBB Meta-model defines the general type SBB_Datacontainer, it is necessary that the specific supplied input and output data to any SBB is specified concretely in a SBBM.

5.1 Interfaces for SBBs

Each SBB consists of an interface which may be used in a system model to apply the functionality of that specific SBB. In this example the input values `data` and `key` are parameters to the `encrypt()` method. This method results in the output data `EncryptedData`. Figure 3 shows the `EncryptionInterface` of the Encryption SBB. More details on interfaces can be found in Section 5.4.

5.2 Preconditions and Postconditions

Describing a SBB only with its input and output values is insufficient in most cases. Therefore two additional artifacts are used to describe so-called Preconditions and Postconditions. Both conditions are optional and should only be used if there are concrete conditions for the described SBB. Using conditions to describe under which circumstances a software component may be executed and what it provides, is based on the concepts of *Design by Contract*. Two interesting works [Mey92, Mey97] describe how these conditions can be applied in software design and development.

A Precondition always describes restrictions, which need to be fulfilled, before the SBB may be applied successfully. Thus they represent a requirement information for the system engineer. In this example, as shown in Figure 3, the SBB_Precondition states that the key used for the encryption must be at least 128 Bit. If the system uses a key that does not meet this precondition the successful application of the SBB is not guaranteed and therefore the postcondition is not provided. In consequence this means that the system engineer is forced to fulfill all the preconditions of any used SBB in order to obtain the postconditions they provide.

On the other hand a SBB_Postcondition describes what the SBB provides if applied successfully. In this example the SBB_Postcondition_Confidentiality states that EncryptedData is now encrypted under a specific key and may only be decrypted if the same key is used. Furthermore it states that an external observer is not able to gain any information about the submitted original data.

If a system modeler now uses EncryptedData (e.g. send it to another system component or over a network device), he is assured that no one without the specific key is able to use the submitted data in any way to gain access to its original plain-text content.

5.3 Combining SBBs

It is also possible that SBBs are combined with other SBBs to enhance and encapsulate functionalities. There are different ways to express such a combination. One way is to use domain-specific SFPs to define the combination. If the combination results in another domain-independent security functionality, it can be useful to describe the combination as another (more complex) SBB. In all cases, the combination needs to be be validated or verified by security expert. If detailed enough and if a formal (or operational) semantic is provided, the SBB model can be the foundation for a rigorous verification. The following paragraphs concentrate on the creation of new SBBs by combining existing SBBs.

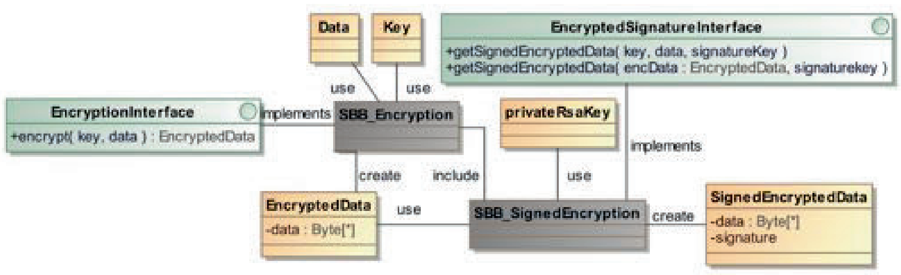


Figure 4: Combining SBBs

Figure 4 shows the case where the output of the Encryption SBB is used in another SBB (SignedEncryption) which generates a signature for that data. The include statements states that the interface for SignedEncryption also accepts the input values from the Encryption SBB. Additionally both SBBs are bound through the `EncryptedData` data-structure. .

Another approach to combine SBBs, as shown in Figure 5, is to use two independent SBBs and include both in an additional SBB. There exist two SBBs, one (Signature) generates a signature of any arbitrary data and the other one (Encryption) encrypts any arbitrary

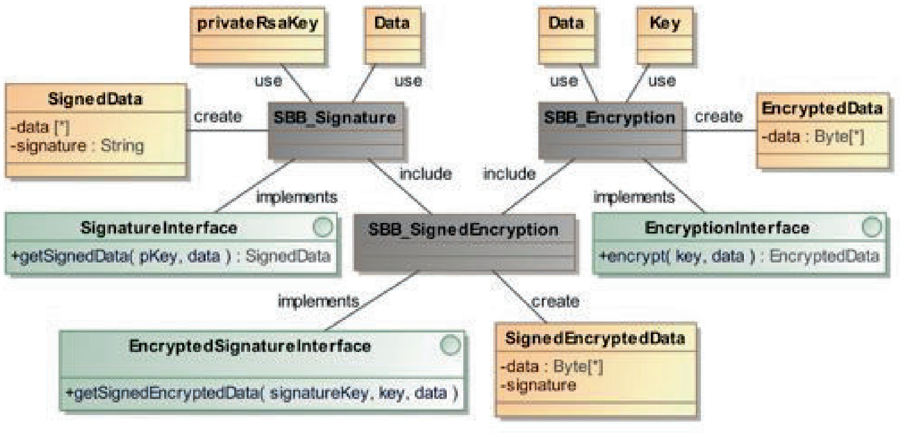


Figure 5: Creating new SBBs by Inclusion

data. Both SBBs may be used independently from another. Both SBBs are included in a third SBB called SignedEncryption. This SBB now uses the SBBs (Signature and Encryption) to generate also an `SignedEncryptedData` data-structure. While in both cases the resulting data-structures are semantically the same, both modeling approaches are different. In the latter case the SignedEncryption SBB is not directly bound to the `EncryptedData` data-structure, which means that `EncryptedData` is no valid input parameter for this SBB by default. (In this example it might be legal to add a separate method which also accepts `EncryptedData` as an input parameter. This decision is left to the SBBM designer and depends on the concrete SBB.)

5.3.1 Aggregation of Conditions

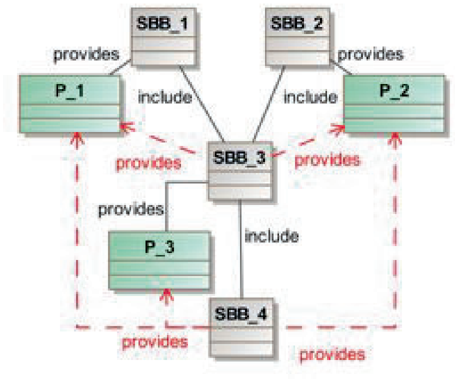
Another important aspect while composing SBBs is that postconditions are aggregated.

Figure 6 shows a general architecture of SBBs and how the aggregation takes place in deeper and nested hierarchical structures.

Whenever a SBB is aggregated of one or more SBBs (SBB_3 includes SBB_1 & SBB_2 and SBB_4 includes SBB_3) the including SBB (aggregate) also provides all postconditions of its included SBBs (parts). The table from Figure 6 shows all SBBs and their provided postconditions, where ”o” marks an indirect aggregated postcondition, ”x” a directly aggregated postcondition and ”-” that no postcondition is adopted.

An aggregate itself may always add additional postconditions, as shown for SBB_3 in Figure 6, but these direct postconditions do not affect the postconditions of its parts.

When working with SBBss in a specific system model, this means that a SBB may be substituted by a SBB which is deeper nested in the same hierarchical aggregated struc-



	P_1	P_2	P_3
SBB_1	x	-	-
SBB_2	-	x	-
SBB_3	o	o	x
SBB_4	o	o	o

Figure 6: Conditional Aggregation

ture (covariant behavior [Car88]). For example SBB_2 may be substituted by SBB_3, but SBB_3 must not be substituted by SBB_1 or SBB_2.

Moreover, it is also possible to substitute a SBB with another independent SBB that is not even in the particular hierarchical aggregation structure (the one of the original SBB). The explanation for this property is that if two distinct and independent SBBs provide the <u>same</u> postcondition, they can be substituted by one another.

For preconditions the same methodology may be used. As long as a particular hierarchical aggregation structure is observed the preconditions of any SBB in that structure stay the same. As soon as a distinct and independent SBB is able to substitute a SBB (by satisfying the original postcondition), the preconditions may be different to any of the preconditions of the substituted SBB. This means that if a SBB provides the same postcondition, but with totally or partially different preconditions, it may be used though.

While composing SBBs it is possible that postconditions are aggregated which contain contradictions. Assuming that the postcondition P_3 is the contrary of P_1 ($P\_3 = !P\_1$). This would mean that SBB_3 has both P_1 and $!P\_1$ as a postcondition.

In general, SBBs provide a process to systematically aggregate postconditions and properties. However, as security is not composable in general it cannot be concluded that the aggregated set of postconditions is actually satisfied for the combined SBB. There is no generic approach to verify these postconditions as this verification strongly depend on the particular properties expressed and on the character of the security functionality represented by the SBB. The SBB expert is responsible to provide evidence that the combination is correct. This evidence can range from best-practice or knowledge of the expert to results of a formal verification.

5.4 Interface from System Model to SBBM

A crucial part in the modeling of SBBs is the definition of an interface. This interface is used either from a system engineer who integrates a SBB into the system model or used to interconnect SBBs within the SBBM itself. The interface is the only visible part and thus the only way to communicate with the SBB. This means that if a SBB is applied in a system model the system modeler may only interact with the specified methods of the building block. While this is the standard procedure how interfaces are used in general, it is a crucial requirement as a SBB expert has to consider the interfaces when designing SBBs.

While a SBB itself provides a solution for a specific security requirement, there may also exist different SBBs solving the same requirement but with different other components involved. As long as the interface of any different SBB is equal, the SBB is easily exchangeable during the development process.

6 Conclusion

The creation of a security model for a system is a very complex task due to the different security requirements, constraints, functionalities, etc. The security engineering process and security building blocks described in this paper helps developers in creating a security model that fulfills all the security requirements of a domain-specific system using security properties and the security building blocks that implement them. Currently, the process is been applied to several use cases of different domains in the SecFutur project, showing good results in each of them.

The SBB Meta-model and the SBB Models as provided in this paper provide one possible approach towards exact specifications of security solutions and their integration into security engineering processes. A validated security solution can be described in a way that preconditions, constraints, dependencies, etc. are exactly expressed and considered in the integration of the SBB into a system.

The next step in this work is to enhance and update the security engineering process with security patterns (that will describe how to create a solution for complex security properties), create certification for the models and increase the DSM online repository with more artifacts. Regarding the SBBs, the next steps are to integrate the concept of SBBs with the SecFutur CSM and DSM and describe domain-specific integrations of SBBs for the realization of more complex security properties.

References

[Car88] Luca Cardelli. A Semantics of Multiple Inheritance. *Information and Computation*, 76:138–164, 1988.

[Gro] Trusted Computing Group. TPM Main specification.

[LSW87] M. Lenz, H.A. Schmid, and P.F. Wolf. Software Reuse through Building Blocks. *Software, IEEE*, 4(4):34–42, july 1987.

[LSW10] H. Lohr, A. R. Sadeghi, and M. Winandy. Patterns for Secure Boot and Secure Storage in Computer Systems. 2010.

[Mey92] B. Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, oct. 1992.

[Mey97] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2. edition, 1997.

[MGS03] Haralambos Mouratidis, Paolo Giorgini, and Markus Schumacher. Security Patterns for Agent Systems. 2003.

[Pea02] S. Pearson. Trusted Computing Platforms, the next security solution. Technical report, HP Labs, 2002.

[RHM11] Jose Fran. Ruiz, Rajesh Harjani, and Antonio Maña. A security-focused engineering process for systems of embedded components. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systemss*, D4RCES '11, pages 4:1–4:9, New York, NY, USA, 2011. ACM.

[Roe01] Schumacher M. Roedig U. Security Engineering with Patterns. In *Pattern Languages of Programs*, 2001.

[SF] *Design of Secure and Energy-efficient Embedded Systems for Future Internet Applications (SECFUTUR), IST-25668, Seventh Framework Programme*. www.secfutur.eu.

[SSS11] Jayraj Signh, Arunesh Singh, and Ms. Raj Shree. Security Patterns in mobile Ad hoc Network: Requirement and Security Management Perspective. 2011.

[Ste06] et al. Steel C. *Core Security Patterns*. Pearson Ed. Inc., 2006.

[SYHJ08] Riccardo Scandariato, Koen Yskout, Thomas Heyman, and Wouter Joosen. Architecting Software with Security Patterns. 2008.